Escuela Politécnica Superior

Aplicación móvil para la gestión de viveros

Máster Universitario en Desarrollo de Software para Dispositivos Móviles

Trabajo Fin de Máster

Autor:
Dennis Alfredo Cayo Guachamín

Tutor:
Miguel Ángel Lozano Ortega

Septiembre 2019

Universitat d'Alacant
Universidad de Alicante

Justificación y objetivos

Esta tesis de máster está enfocada en la gestión agrícola. La aplicación proporciona herramientas que ayudan en la gestión de tareas del campo y de personal. Además, tanto los empleados como el personal encargado podrán controlar el trabajo realizado o las tareas pendientes desde su smartphone en cualquier lugar y a cualquier hora.

Como el fin es lograr un instrumento de trabajo colaborativo, se ha optado por utilizar una aplicación web, una aplicación móvil en Android y un servidor donde estará almacenada toda la información y lógica de los servicios web, que son consultados tanto de la aplicación web como desde la aplicación móvil para la obtención y manipulación de los datos. Estas aplicaciones contarán con una interfaz agradable y sencilla al usuario final, con el objetivo de que el usuario pueda introducir información, realizar consultas.... Independientemente de la localización de cada empleado.

Para conseguir dicho fin, se realizará un análisis de mercado y un estudio previo de la gestión agrícola con el objetivo de brindar al usuario una experiencia agradable, que cumpla con sus expectativas, usando técnicas de la ingeniería informática y el sector agrícola.

Los objetivos que se persiguen son:

- Función multi-dispositivo para trabajar con tu equipo de forma simultánea.
- Tener todos los datos centralizados y accesibles al instante.
- Gestión de personal.
- Gestión de facturas y albaranes.
- Gestión de trabajos de los usuarios.

Agradecimientos

Familia y Noemí Martínez Machado.

Citas

Todo lo puedo en Cristo que me fortalece.

Filipenses 4:13(NVI)

Dedicatoria

Índice

Índice de ilustraciones

1 Introducción

En la actualidad el mundo junto con la tecnología está cambiando para adaptarse a los usuarios. Existen multitud de áreas donde podemos hacer uso de nuevas tecnologías con el fin de realizar un proceso de automatización y realizarlas de la manera más óptima posible. Nosotros vamos a focalizar nuestro esfuerzo en el sector agrícola. Se trata de un proyecto real y que suplirá las necesidades de un cliente.

Los cambios en el sector agrícola lo están reflejando. Desde hace años la agricultura ha sufrido múltiples cambios y transformaciones que dejan atrás el arado de la tierra o las tradicionales tareas manuales para dar la bienvenida a un mundo de actividades que requieren cada vez menos esfuerzo, pero también logrando mejores resultados. ¿Cómo es posible? Ésta es la pregunta que muchos de ustedes se estarán haciendo. Hablamos de un antes y un después, de dos realidades diferentes pero complementarias. El control y la automatización han entrado en escena.

Existen múltiples áreas donde podemos hacer uso de las nuevas tecnologías con el fin de realizar un proceso de automatización y hacerlo de la manera más eficiente.

Nuestra idea nace a partir de la necesidad real de un cliente, que ha analizado varios softwares agrícolas, en los que ha encontrado varios problemas como:

- No cubrían sus necesidades básicas.
- Más funcionalidad de la necesitada.
- Complejidad en el uso.
- El coste y mantenimiento tenían un precio elevado.

El objetivo principal es centrarnos en esta área y cubrir las necesidades de nuestro cliente: facilitar la gestión de trabajos realizados, control de personal, pago de facturas, control sobre maquinaria y otras actividades. Con ello se busca olvidar la tradicional gestión en papel, en definitiva, “olvidarse del papeleo”.

El usuario podrá tener todo en su móvil, llevando un control y la explotación bien organizada desde cualquier sitio y en cualquier momento de forma instantánea.

El objetivo de este trabajo de fin de Máster es el desarrollo de una aplicación móvil nativa en Android para el control y gestión de un vivero. Se contará con un servidor que ofrecerá una serie de servicios REST a través de los cuales la aplicación móvil accederá a una base de datos centralizada. Y para ampliar la gestión contará con una aplicación web.

La aplicación llevará un control del personal, de la maquinaria, del stock, de los proveedores y de las horas de trabajo. La aplicación estará adaptada a dos perfiles: gestores y trabajadores, de acuerdo con el rol establecido se podrán realizar una serie de operaciones.

Se plantea el problema de automatizar la gestión de procesos en el sector de la agricultura, en esta gestión entra diversos factores:

- La gestión del personal, como pagos, control de horarios, trabajos pendientes, etc.
- La gestión interna de la empresa
- Asignar una tarea concreta a un empleado
- Terminar una tarea y firma de ésta
- Crear nuevas tareas
- Asignar una tarea concreta a un empleado
- Terminar una tarea y firma de ésta
- Estimación de un trabajo basado en las horas empleadas y materiales, problemática obtenida con el fin de mejorar la ejecución de dicha tarea
- Gestión de personal, como el pago de salarios
- Gestión de albaranes y facturas
- Gestión y control de maquinarias

2 Estado del arte

Vamos a realizar un estudio de mercado de diferentes aplicaciones del sector agrario, estudiando sus características, sus ventajas y qué es lo que ofrece al usuario, es decir, el propósito que persigue; como por ejemplo ayudar en la gestión del trabajo realizado por sus empleados u otro tipo de tareas.

Hay varias preguntas que deberíamos hacernos con respecto al software agrario:

- ¿Qué es un software agrícola?
- ¿Cómo funciona?
- ¿Cuáles son las principales funciones?
- ¿A quién va dirigido este tipo de software?

Y la pregunta más importante puede ser

- ¿Por qué integrar un software en la gestión agrícola, qué objetivos persigue su integración?
  - Ventajas
  - Desventajas
  - Funcionalidades

Vamos a ir dando respuesta a cada una de estas preguntas y posteriormente realizar una comparativa entre distintos tipos de softwares agrícolas.

2.1 ¿Qué es un software agrícola?

Como punto de partida podemos decir que el objetivo principal es hacer más fácil el trabajo diario en el campo, de aquí la importancia de tener una herramienta con la que poder registrar y gestionar la producción, las tareas, los costes entre otras funcionalidades.

Como definición podemos decir que es un software de planificación de recursos empresariales pensado para facilitar la gestión del trabajo en el campo. La mayoría de estos programas y aplicaciones tiene como labor principal

optimizar múltiples tareas directamente relacionadas con este oficio. Así mismo ayuda a los agricultores a gestionar el trabajo en el campo y los procesos de negocios.

2.2 ¿Cómo funciona el software agrícola?

Es muy simple, puede ser desde una aplicación web que disponga de una conexión a internet donde tendremos acceso a la información (desde un ordenador, una tablet o un *smartphone*), hasta una aplicación móvil como complemento a la aplicación web. De esta forma podremos acceder a los datos en tiempo real desde cualquier lugar sin importar la localización.

2.3 ¿Cuáles son las principales funciones de un software agrícola?

Estas aplicaciones te permiten aumentar la productividad, rentabilidad y facilitar la gestión del negocio. A continuación, podemos observar algunas de las funcionalidades:

- Almacenar información sobre la actividad.
- Consultar datos en tiempo real y desde cualquier situación geográfica.
- Toda la información accesible en un mismo sitio.
- Poder utilizar tu cuaderno de campo en cualquier momento y lugar.
- Gestionar insumos y controlar la cadena de suministros (control de inventarios).
- Calcular costes.
- Tener un mapa satelital preciso.
- Predecir resultados.
- Obtener estadísticas del campo y como evitar perdidas.
- Automatizar procesos de negocios.
- Gestión administrativa: pedidos, facturación y tesorería.
- Control técnico y trazabilidad de los cultivos.

2.4 ¿A quién va dirigido este tipo de software agrícola?

Va destinado a los agricultores y a los profesionales del sector forestal, algunas de ellos ya han implementado esta herramienta de trabajo. Toda empresa que opera en el sector agropecuario es susceptible de poder optimizar su actividad, gracias al desarrollo y a la adaptación de las nuevas tecnologías al sector de la agricultura.

2.5 ¿Por qué integrar un software en la gestión agrícola, qué objetivos persigue su integración?

La integración de un software trae consigo una serie de ventajas, pero también puedo acarrear una serie de desventajas.

Como respuesta a la pregunta planteada, de porqué puede ser beneficioso incorporar un software agrícola a nuestro negocio; es porque permite que las infraestructuras estén centralizadas, es decir, unificar las actividades en un solo lugar y poder tener una visión global de la evolución de la empresa. Además, permite ahorrar tiempo en la planificación de la producción. De esta forma se pueden obtener estadísticas y predecir resultados. Además, hay que destacar que permite llevar tu cuaderno de explotación agrícola a todas partes.

2.6 Comparativa

Si escribimos en *Google* "software agrícola" podremos observar que existen varios tipos de software cuando se trata de este tema. Entre las aplicaciones más buscadas generalmente por los usuarios son las de gestión de las granjas, viveros y/o las de gestión forestal (silvicultura). En nuestro caso haremos más hincapié en la gestión de la granja y viveros (control de personal, facturas, gestión de pagos, etc.)

Entre los programas más utilizados en el sector de la agricultura podemos nombrar algunas App como *AgroMe, AgroFrutex, Sismagro, aHora, Farmlogs, Sismagro, Agroptima,* entre otras. Estos programas son sumamente completos, asisten en casi todas las áreas del trabajo en el campo desde mapeo satelital, seguimiento de cultivos, de plagas y de condiciones meteorológicas hasta la gestión de stocks, de contactos y tareas administrativas.

Cada uno de los programas tienen sus funcionalidades, así como ventajas y desventajas que lo pueden hacer más adecuado acorde con unas necesidades principales.

En los siguientes apartados vamos a explicar características de cada uno de ellos. Se terminará haciendo una comparativa con nuestra aplicación desarrollada, con el fin de explicar porqué la necesidad de ésta y no haber utilizado una de las aplicaciones/softwares ya existentes en el mercado.

2.6.1 Agrivi

Agrivi es un software de *Farm Management Software* creado por la empresa *Agrivi* con sede en Estados Unidos. El precio de este programa es de 19$/mes.

Agrivi es un software de gestión agrícola basado en el conocimiento con el fin de ayudar a los agricultores en la toma de decisiones basada en un histórico de datos para mejorar la productividad y la rentabilidad, es decir, se crean estadísticas que puedan prever casos futuros. Agrivi facilita a los agricultores el poder gestionar todas las actividades del campo, de finanzas, de inventario, de mano de obra y permite obtener información sobre el rendimiento global de la agricultura con un solo clic. Construido partiendo de una base de conocimientos agrícolas, plagas y enfermedades.

2.6.2 Agrimap

Agrimap es un Software de Farm Management Software creado por la empresa Agrimap con sede en Estados Unidos. El precio de este software es de 30$/mes.

Agrimap se caracteriza por una filosofía diferente acerca de la tecnología en la agricultura. Su lema es "creemos en un mundo donde los agricultores pueden capturar fácilmente sus datos, compartir y aprender unos de otros".

Ellos afirman que muchos usuarios de otras aplicaciones se ven perdidos o no saben hacer uso de una aplicación cuando se presenta con un sistema complejo, tiene poca usabilidad o tiene una interfaz con muchas funcionalidades.

Su diseño simple y estético permite la adopción de la tecnología en cualquier negocio agrícola. Agrimap permite a los agricultores tener el control del entorno agrícola, de sus campos, del cuaderno de campo y la gestión de personal.

2.6.3 Cropio

Cropio es un software de *Farm Management Software* creado por la empresa *N.S.T. New Science Technologies* con sede en Estados Unidos. El precio de este software es de 100$/año.

Cropio es un sistema de gestión del campo que facilita la monitorización remota de las tierras agrícolas y permite a sus usuarios planificar y llevar a cabo las operaciones agrícolas de manera más eficiente y en tiempo real. Cropio proporciona actualizaciones en tiempo real sobre las condiciones del campo, determina los niveles de vegetación e identifica las áreas problemáticas, ofrece previsiones meteorológicas precisas y una visión general real del mercado de materias primas agrícolas.

2.6.4 Agroptima

En su web oficial encontramos la siguiente definición:

"Agroptima es un software agrícola en la nube (puedes trabajar con él desde dónde quieras y cuándo quieras) que permite al agricultor llevar de forma fácil e intuitiva la gestión agronómica y económica de su explotación agrícola y a cumplir con las leyes agrícolas. Alíate con la mejor tecnología para tu día a día en el campo."

[Referencia 4]

Agroptima nos ofrece una serie de funcionalidades que hacen que sea una aplicación bastante competitiva en el mercado. A continuación, vamos a exponer sus características:

- **Gestión de labores agrícolas:** Anota tus labores desde el campo y ten tus datos siempre a tu alcance.

- **Geolocalización de parcelas:** Dibuja o importa tus parcelas y píntalas según cultivos, variedades, etc.
- **Información de fitosanitarios, semillas, abonos:** Base de datos del MAPAMA con caducidad de productos.
- **Generación de Cuaderno de Campo y de Fertilizantes:** En solo un clic, tendrás tus Cuadernos Oficiales listos para presentar.
- **Control de stock:** Sabrás en todo momento que tienes en tus almacenes para poder llevar un control y una buena planificación.
- **Control de costes e ingresos:** Introduce tus precios y Agroptima calcula automáticamente los costes de tu explotación agrícola.
- **Trabaja con o sin Internet:** La aplicación móvil de Agroptima funciona sin necesidad de estar conectada a Internet

2.6.5 Ahora

Su sistema de información está enfocado a la gestión y la mejora de la productividad de las compañías, ofreciendo un gran número de funcionalidades. Por poner algún ejemplo, con nuestro software para empresas agrícolas, podrá generar diariamente como tarifas, marcando únicamente un precio final. Asimismo, podrá gestionar la recolección, los productos o las transformaciones, entre otras muchas cosas, éstas son algunas de las características que ofrece:

- **Generación diaria de tarifas:** permite establecer de manera diaria el precio de venta de los productos de manera ágil, indicando únicamente un precio final directo o un porcentaje de beneficio.
- **Gestión de productos:** podrá dividir y organizar sus productos según familias (especie, grupo varietal, variedad agronómica, etcétera.)
- **Gestión de transformaciones:** gestión sencilla de envasados basados en lote y lleve la trazabilidad de aquellos productos que hayan sido comprados a granel y posteriormente envasados para su venta.

- **Tratamientos fitosanitarios:** control los productos aplicados en cada fecha y hora. Asimismo, registre la evaluación de los resultados y el consumo del stock de la cantidad de producto que se utilice en cada tratamiento.

- **Gestión de partidas de producción:** crear productos con lotes, así como generar partes de materiales y planificar tareas. Del mismo modo, es posible trabajar con las partidas por años de venta y de plantación.

- Gestión de Fincas Propias

- Gestión Fincas Ajenas

2.7 Nuestro caso concreto

La pregunta que se puede hacer es ¿por qué desarrollar una aplicación agrícola si ya existe varias en el mercado que cuentan con un gran rodaje y una gran cantidad de usuarios?

Tanto la aplicación móvil como la aplicación web se basa en una necesidad real de un cliente. Esta persona se dedica al sector de la agricultura, posee varios campos y tiene a su cargo una gran numero de empleados. Él nos cuenta que ha acudido a varias ferias de agricultura y gente experta en el tema le ha hecho varias demostraciones de programas con el fin de poder cubrir sus necesidades. Pero nos afirma que hay muchos programas que no cubren sus necesidades principales o por el contrario hacen más de lo que él necesita, siendo éstos complejos para su uso.

Incluso nos cuenta que ha estado investigando. Muchos de los programas que ha encontrado, incluyendo demos no se adaptan a lo que él necesita. Por lo que le surgió la duda de porqué no desarrollar su propio sistema, es decir, su propia aplicación que dé respuesta a sus necesidades. Es de allí de donde nace la idea de desarrollar este proyecto.

Nuestro proyecto cuenta con varias funcionalidades:

- **Aplicación web**, que ha sido desarrollada para realizar una serie concreta de funcionalidades

- **Aplicación móvil** donde se implementarán una serie de funcionalidades concretas, siempre como complemento a la aplicación web.

- **Servidor web** basado en NodeJs con express, con una **base de datos** desarrollada en PostgreSQL

2.8 SafeRefuge basado en una necesidad real

Nuestro trabajo principal ha sido el desarrollo de la aplicación móvil basada en Android. Desde esta aplicación se podrán realizar una serie de tareas y funcionalidades especificas. A continuación, vamos a explicar algunas de las funcionalidades, no entraremos mucho detalle debido a que las explicaremos en próximos apartados con más detenimiento.

Estas son algunas de las funcionalidades:

- Gestión de facturas.
- Gestión de albaranes.
- Realizar pagos.
- Gestión de personal.
- Gestionar trabajos realizado.

3 Objetivos

Una vez expuestas las necesidades y las posibles carencias en el software ya existente en el mercado se ha planteado crear un software propio que realiza una serie tarea adecuadas y adaptadas a unas necesidades concretas. De esta forma podemos establecer como objetivo de este trabajo de fin de Máster el desarrollo de una aplicación móvil nativa en Android para el control y gestión de un vivero. Se contará con un servidor que ofrecerá una serie de servicios REST a través de los cuales la aplicación móvil accederá a una base de datos centralizada. La aplicación llevará un control del personal, maquinaria, stock, proveedores, y horas de trabajo. La aplicación estará adaptada a dos perfiles: gestores y trabajadores, de acuerdo con el rol establecido se podrán realizar una serie de operaciones.

Se plantea el problema de automatizar la gestión de procesos en el sector de la agricultura, en esta gestión entra diversos factores:

Con el rol de administrador

- La gestión del personal, control de horarios
- Trabajos realizados, pendientes de terminar, etc.
- La gestión internada de la empresa
- Gestión de facturas, albaranes.
- Gestión e histórico de pagos.

Con el rol de usuario:

- Poder ver los trabajos pendientes o realizados
- Ver histórico de pagos.

Nuestro principal objetivo es el desarrollo de la aplicación móvil que discriminará entre los usuarios y el usuario administrador. Cada uno podrá realizar una serie de acciones según su rol especificadas anteriormente. Además, la aplicación cuenta con su propia base de datos de forma que se podrán acceder a los datos sin necesidad de tener acceso a internet, y cuando dispongamos de

internet los datos se actualizarán, de esta forma garantizamos que la aplicación disponga de todos los datos actualizados.

Aparte de la gestión realizada con la aplicación móvil, también se dispone de una aplicación web que amplía la gestión.

4 Tecnologías usadas

Para el desarrollo del proyecto se ha hecho un análisis previo de las tecnologías que podemos hacer uso y se ha optado por usar tecnologías con las que se hayan trabajado previamente. Así, evitamos la dificultad de aprender nuevas tecnologías y sobretodo el tiempo invertido. De eta forma nos podemos centrar en lo verdaderamente importante que es desarrollo del proyecto y la implementación.

En los siguientes apartados vamos a ir desarrollando las tecnologías que sea han usado para el proyecto. Queremos recalcar que la tarea de este trabajo de fin de máster es centrarse en el desarrollo de la aplicación móvil en Android, pero es importante explicar el contexto del proyecto para tener una visión genérica del mismo.

4.1 Heroku

Para el desarrollo de aplicaciones, hay que ser consciente del impacto que pueda tener cuando se empieza a tener una cierta cantidad de usuarios, por esto muy importante lanzar la aplicación sin tener complicaciones de infraestructura, administración de servidores, las bases de datos y la seguridad que estos deben de tener entre otras cosas.

Actualmente en el mercado existen una serie de plataformas conocidas como PaaS en sus siglas en inglés *"Platform as a Service"* o en español *"Plataformas como Servicios"* que, además de ser la evolución de las IaaS (Infraestructura como Servicio).

Un ejemplo lo encontramos en el EC2 de Amazon (Amazon Web Services) donde te proporcionan un servidor y el usuario se encarga de provisionarlos y administrarlos con PaaS, de esta forma el usuario se olvida de todas las cuestiones de administración, seguridad, etc. Pues se utiliza directamente una plataforma que lo hace por ti.

4.1.1 ¿Qué es *Heroku*?

En su web oficial la siguiente la siguiente definición:

Heroku is a cloud platform that lets companies build, deliver, monitor and scale apps — we're the fastest way to go from idea to URL, bypassing all those infrastructure headaches.

[Referencia 23]

Heroku es uno de los *PaaS* más utilizados en la actualidad, gira en torno al ámbito empresarial por su fuerte enfoque en resolver el despliegue de una aplicación. Además, permite manejar los servidores y sus configuraciones de maneara muy sencilla, escalamiento y la administración.

Cabe destacar que es una plataforma en la nube, lo que significa que a los desarrolladores no deben preocuparse por la infraestructura, sino que solamente nos tenemos que centrar el esfuerzo del desarrollo de la aplicación, lo que evita todos los problemas que puede suponer llevar nuestra una idea a un servidor

Heroku a diferencia de otras plataformas permite desarrollar prácticamente con cualquier lenguaje de programación:

- Ruby
- Java
- PHP
- Go
- NodeJS, etc.

En la documentación podemos ver todos los lenguajes que soporta.

En resumen, a Heroku solo le dices qué lenguaje de backend estás utilizando o qué base de datos vas a utilizar y te preocupas únicamente por el desarrollo de tu aplicación.

4.1.2 Pricing

Para usar la aplicación es obligatorio el registro.

En cuanto al precio, difiere un poco de las necesidades que tenga un usuario en concreto. Hay varios tipos de registro, a continuación, procedemos a explicarlos

- **Registro gratuito**: mediante el registro gratuito podemos acceder a las prestaciones básicas, pero no podremos disfrutar de todas las posibilidades que ofrece esta plataforma. Por ejemplo, tenemos unas ciertas horas de uso al mes de los servicios, otra restricción en cuanto a uso es cuando el servidor no es uso por algún tiempo, este se pone en modo internación y hay que activarlo de manera manual

- **Registro con pago**: el registro pagando nos permite acceder a todas las prestaciones de HEROKU. Servicios siempre activos y otras opciones

- Otra posibilidad es registrarse de forma gratuita y solo pagar por aquellas prestaciones que sean necesarias, es decir, depende de los servicios usados se paga una cantidad concreta.

4.1.3 Características

Los Dynos son piezas fundamentales del modelo de arquitectura de Heroku, son las unidades que proveen capacidad de cómputo dentro de la plataforma. Están basados en Contenedores Linux.

Cada Dyno está aislado del resto, por lo que los comandos que se ejecutan y los archivos que se almacenan en un Dyno, no afectan a los otros. Además, cada Dyno provee el ambiente requerido por las aplicaciones para ser ejecutadas.

- Elasticidad y crecimiento. La cantidad de Dynos asignados a una aplicación se puede cambiar en cualquier momento a través de la línea de comandos o el dashboard.

- Tamaño. Heroku ofrece diferentes tipos de dynos, cada uno con diferentes capacidades de procesamiento y memoria.

- Routing. Internamente los routers realizan un seguimiento de la ubicación de los Dynos que estén corriendo, y redirigen el tráfico de acuerdo a la misma.

- Seguimiento. Existe un manejador de Dynos, el cual monitorea de forma continua los dynos que se estén ejecutando. En caso de una falla en un Dyno, este es eliminado y creado nuevamente.

- Distribución y redundancia. Los Dynos se encuentran aislados uno de otro. Esto implica que, de existir fallos en la infraestructura interna de alguno de ellos, los otros dynos no se ven afectados, y consecuentemente tampoco la aplicación.

[Referencia 9]

4.1.4 Ventajas

Las ventajas que nos ofrece Heroku son:

- Es gratuito para aplicaciones que tengan poco consumo

- Es poliglota, es decir, permite el uso de diferentes lenguajes de programación.

- Es una plataforma muy sencilla de usar. Además, posee una extensa documentación
- Posibilidad de integrar varios servicios dentro de su estructura.
- Las actualizaciones en Heroku no afectan a nuestra plataforma informática.
- Con tener acceso a internet se puede acceder desde cualquier lugar y dispositivo compatible con la computación en la nube.

4.1.5 *Heroku*, uso en nuestra aplicación

Para desplegar nuestra aplicación en heroku, para la parte del servidor hemos optado por usar node.js con express y para la parte de la base de datos se usa postgreSQL.

Estos son add-ons propios, es decir, módulos de heroku. Su instalación y configuración es muy simple.

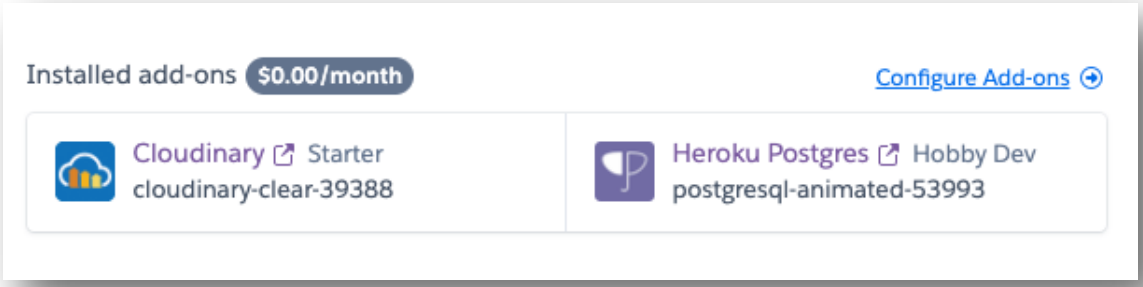


Ilustración 1. Módulos instalados en Heroku

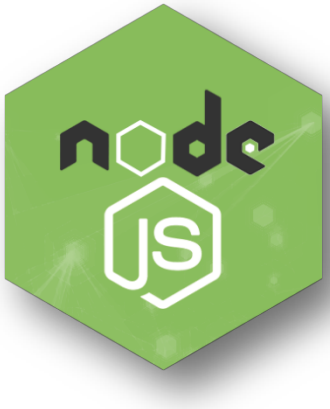
Para hacer uso de nuestra API la podemos encontrar en el siguiente enlace.

- https://safe-refuge-74613.herokuapp.com/api/

Tanto las aplicaciones móviles como las aplicaciones web acceden a la misma dirección para realizar las peticiones o acciones correspondientes.

En los siguientes apartados vamos a describir de forma más extensa la configuración y como se han implementado cada uno de los módulos anteriores. Así mismo facilitaremos algún ejemplo de llamadas a nuestro api y la respuesta devuelta.

4.2 Servidor en *Node.js* con *Express*



Node.js una herramienta que ha alcanzado una gran popularidad en los últimos tiempos, hasta el punto de convertirse en un componente indispensable en el desarrollo de aplicaciones Web. Nuestro objetivo no es dar una definición extensa de lo que es Node.js, sino dar a conocer las características por lo cual lo hemos elegido para nuestro desarrollo.

Antes de empezar vamos a indicar que Node.js es Open Source, cuenta con una extensa comunidad de usuarios y colaboradores, y que está disponible a través de su página web https://nodejs.org/es/

4.2.1 ¿Qué es *Node.js*?

Es difícil de explicar de forma rápida, no es tan simple como decir que es un servidor web o un entorno de desarrollo, porque depende el enfoque que le de un usuario puede estar orientado a que cumpla diversas funcionalidades. Bien vamos a partir con la definición que nos facilita Wikipedia:

"Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web.4 Fue creado por Ryan Dahl en 2009 y su evolución está apadrinada por la empresa Joyent, que además tiene contratado a Dahl en plantilla."

[Referencia 29]

Node.js es una librería y entorno de ejecución de E/S dirigida por eventos y por lo tanto asíncrona que se ejecuta sobre el intérprete de JavaScript creado por Google. Node.js es una forma de ejecutar JavaScript en el servidor, además de mucho más es un entorno Javascript del lado del servidor que es capaz de ejecutar javascript utilizando el motor V8, desarrollado por Google para uso de su navegador Chrome. De esta forma aprovecha el motor V8 permite a Node

proporciona un entorno de ejecución del lado del servidor que compila y ejecuta javascript a velocidades increíbles

4.2.2 ¿Cómo funciona?

Para explicar el funcionamiento de node.js vamos a exponer un pequeño ejemplo comparando con otro sistema

En lenguajes como PHP o java cada vez que se genera una conexión en el servidor se crea un nuevo hilo, en un servidor con las características siguientes.

- 4 GB de RAM

Si cada conexión necesita 2 MB, como mucho se podrían atender 2000 conexiones simultaneas. Esto supone un problema grave ya que a medida que crece el numero de usuarios, si se desea que la aplicación soporte más usuarios, necesitará agregar más y más servidores. Lo que supone un coste extra. Otra problemática a nivel técnico puede ser que un usuario este usando diferentes servidores para cada solicitud, así que cualquier recurso compartido debe almacenarse en todos los servidores. Esto puede generar un cuello de botella en toda la arquitectura de la aplicación web debido las respuestas del servidor está limitada al número máximo de conexiones concurrentes que puede manejar un servidor.

Node.js resuelve este problema debido a la forma en que se realiza una conexión con el servidor. En node.js cada conexión dispara una ejecución de evento dentro del proceso del motor de en lugar de generar un nuevo hilo de OS para cada conexión y por lo consiguiente la asignación de memoria.

Otro punto a favor es que se afirma que nunca se quedará en punto muerto, porque no se permiten bloqueos y porque no se bloquea directamente para llamados E/S. Por lo que se puede afirmar que un servidor que ejecute node.js puede soportar decenas de miles de conexiones concurrentes.

4.2.3 Características de *node.js*

Uno de los motivos para haber porque se usa JavaScript es que es un lenguaje extendido y utilizado en el desarrollo Web. Por este motivo, la curva de aprendizaje resulta sencilla para un gran número de desarrolladores. Por otro lado, las características de asincronía y diseño orientado a eventos de JavaScript lo hacían apropiado para el propósito principal de Node.js.

A continuación, vamos a explicar algunas características de node.js:

4.2.3.1 Escalabilidad

Node se creo con un propósito, que es la capacidad de soportar una gran cantidad de conexiones simultaneas a un servidor. Node.js emplea un único hilo y un bucle de eventos asíncrono por lo que las nuevas peticiones son tratadas como eventos en este bucle.

Este es el motivo por el que las características asíncronas y los eventos de JavaScript encajan tan bien en la filosofía de Node.js, porque permite que sea capaz de gestionar múltiples conexiones y peticiones de forma muy eficiente y con gran rapidez.

4.2.3.2 Node package manager

Otro de los motivos del éxito de Node.js es su gestor de paquetes NPM (Node Package Manager). Este nos permite acceder a una enorme cantidad de librerías Open Source desarrolladas por la comunidad

Entre los miles de paquetes disponibles, tenemos varios destacables expuestos a continuación:

- Express,
- React,
- Gulp,
- Socket.Io,
- Jade,
- Mongoose,
- Browserify
- Forever
- Y otras muchas más.

En nuestro caso hemos usado node.js en combinación con **Express**.

4.3 Express.js

Espress.js es sin duda el framework por excelencia y más conocido de node.js. Para dar una definición más exacta según sus creadores:

Expres.js es un framework de desarrollo de aplicaciones web minimalista y flexible para Node.js". Está inspirado en Sinatra, además es robusto, rápido, flexible y muy simple.

[Referencia 31]

Entre otras características que ofrece podemos destacar:

- Router de URL (Get, Post, Put …)
- Facilidades para motores de plantillas (Jade, EJS…)
- 11 Middleware via Connect.
- Un buen test coverage.
- Session Handler.

4.4 Base de datos con *PostgreSQL*

PostgreSQL en la actualidad es uno de los gestores de bases de datos más usados del momento.

4.4.1 ¿Qué es *PostgreSQL*?

PostgreSQL es una de las opciones más interesantes en bases de datos relacionales open-source, es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto. Michael Stonebraker inició el proyecto bajo el nombre Post Ingres a mediados de los 80's con la idea de solucionar problemas existentes en las bases de datos en esa época. Como muchos de los proyectos de código abierto, *PostgreSQL* no es propiedad de una empresa o persona, sino que es dirigido por una comunidad de desarrolladores.

4.4.2 Características de *PostgreSQL*

- **Es orientado a objetos**, es decir, todos los elementos de nuestra base de datos van a poder tratarse como objetos, algo parecido a un lenguaje de programación.

- **Es multi-sistema**: por lo tanto, puede ser instalado en diversos sistemas operativos como *Microsoft Windows, GNU/Linux, MacOS, BSD* y muchos otros más.

- **Es extensible**: podemos añadir funcionalidades extra Podemos encontrar documentación de como extender su funcionalidad

- **Es escalable**: es capaz de manejar bases de datos enormes, de más de 100 Terabytes y funciona bajo licencia libre. Puede ser usarlo para cualquier propósito sin ningún tipo de restricción ni problema.

4.4.3 Usos de *PostgreSQL*

PostgreSQL se utiliza en diversos ámbitos, aquí te dejamos algunos de los ejemplos donde se hace uso de este gestor de bases de datos:

- Almacenamiento de datos (DWH).

- En servicios como Amazon Web Services Redshift.
- Para procesado de datos, almacenado tanto en la propia instancia como en otros servicios que puedan conectarse.
- En sistemas de información geográfica, como el servicio de mapas web o también en servicios móviles OpenStreetMap.
- En bases de datos para servicios web.
- En CMS como Drupal o WordPress.
- En la conocida base de datos de cine IMDb.

[Referencia 22]

PostgreSQL también es usada en una plataforma como servicio: Algunos ejemplos que lo utilizan, es Skype, que lo usa para dar servicio a todos sus departamentos y diferentes aplicativos. Y muchas otras empresas como:

- Telefónica,
- BBVA,
- Petrobras,
- el metro de Sao Paulo,
- el servicio de información meteorológica del Reino Unido.
- Entre otras muchas más.

Debido a la popularidad que PostgreSQL ha cogido en los últimos tiempos, se ha convertido en uno de los gestores de bases de datos de código libre más potentes y fiables del mercado. PostgreSQL encaja en nuestro proyecto ya que podemos ver al usarlo obtenemos seguridad, confiabilidad, estabilidad y la oportunidad de mejorar y aportar desarrollos funcionales para hacer un manejador de base de datos aún más óptimo.

En el siguiente apartado se muestran las tablas definidas para nuestra aplicación. Algunas tablas son usadas solo desde la aplicación web y otras son exclusivas de la aplicación móvil.

4.5 Aplicación móvil

En este aparado vamos a explicar las tecnologías usadas para el desarrollo de nuestra aplicación, no haremos mucho hincapié ya que en apartados posteriores se explicarán más en detalle como parte del desarrollo.

4.5.1 Room

Room es una librería que abstrae el uso de *SQLite* al implementar una capa intermedia entre esta base de datos y el resto de la aplicación. De esta forma se evitan los problemas de *SQLite* sin perder las ventajas de su uso.

La aplicación de dispone de una propia base de datos, basada en *Room*. Nos explicaremos ¿Qué es una base de datos Room en *Android*?

Pues la respeta es sencilla, de forma coloquial podemos decir que:

- ***SQLite*** es un software que se utiliza (aparte de para otras muchas aplicaciones) para crear bases de datos en aplicaciones para Android. Ya sabes: cierras una aplicación, la vuelves a abrir, y los datos que introdujiste siguen ahí.

- ***Room***, presentado en la conferencia Google I/O de 2017, simplifica el uso de *SQLite* (como verás a continuación).

Room funciona con una arquitectura cuyas clases se marcan con anotaciones preestablecidas. Por otro lado, la mayoría de las consultas a la base de datos sí se comprueban en tiempo de compilación.

Las partes de las que se compone Room son las siguientes:

- ***Entity***: son clases que definen las tablas de la base de datos y de las entidades a utilizar.

- ***DAO***: interfaces que definen los métodos utilizados para acceder a la base de datos.

- ***RoomDatabase***: sirve de acceso a la base de datos *SQLite* a través de los DAOs definidos.

5 Especificación y diseño

Nuestro proyecto este compuesto por varios sistemas, entre ellos:

- Servidor en *NodeJs* con *express*
- Base de datos con *postgreSQL*
- Servidor en firebase para las notificaciones push
- Aplicación web
- Aplicación móvil para *Android*
- Aplicación móvil para *iOS*

5.1 Diagrama de la arquitectura del sistema

En la siguiente figura podemos observar cual va a ser la estructura de nuestra aplicación, por un lado, tenemos el servidor en *NodeJs* con el que la aplicación web como las aplicaciones móviles van a interactuar para la petición de datos al servidor. Y la base de datos en *postgreSQL* donde se almacenará toda la información que generan ambas aplicaciones.

Por otro lado, tenemos el servidor de firebase, esto cabe mencionar que todavía no esta implementado en nuestro sistema, forma parte de la estructura del proyecto, pero es una idea futura que se desarrollara. Esto es debido a que queremos tener una primera versión del sistema funcional y que el desarrollo del proyecto se realice de forma gradual

La idea que se tiene es que cualquier acción o evento generado tanto en la web como en los móviles sea notificado a un usuario concreto.

Un ejemplo podría ser en la creación de un trabajo por parte del administrador del sistema, pues que esta acción sea notificada el usuario con el siguiente mensaje:

"Tiene un trabajo que realizar durante el día de hoy"

"Se te ha realizado un abono en tu cuenta bancaria"

Y que el usuario entre en a la aplicación para ver más detalles o tener más información, como los materiales que debe usar para realizar el trabajo o cuando

debe iniciar el trabajo. O ver la cantidad que se le ha abanado en su cuenta corriente y poder verificar si el pago es correcto.

Esto es una idea que se desarrollarla en un futuro cuando tengamos primeramente todas funcionalidades desarrolladas en ambos partes (web como móvil).

En el siguiente diagrama podemos ver la estructura de nuestra aplicación, y como está distribuida.

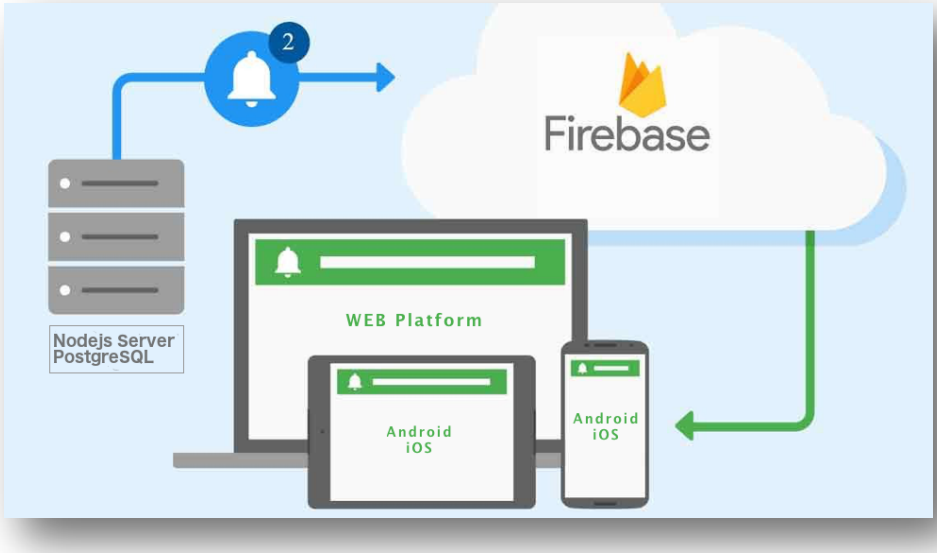


Ilustración 2. Arquitectura del sistema

5.2 Base de datos

En el apartado de tecnologías usadas se explicó que es ***postgreSQL*** y porque lo hemos elegido como nuestro sistema gestor de bases de datos de habiendo tantos en el mercado, es capaz de funcionar de manera estable en el servidor y, por lo tanto, resulta bastante robusto, esto es una de las principales características que se persigue. Además, es consistente y tolerante a fallos. Es compatible con el modelo relacional, ya que asegura siempre su integridad referencial.

A modo de resumen vamos a citar algunas de las características principales:

- Alta concurrencia.
- Soporte para múltiples tipos de datos de manera nativa.

- Soporte a triggers.
- Trabajo con vistas.
- Objeto-relaciona.
- Soporte para bases de datos distribuidas.
- Soporte para gran cantidad de lenguajes.

Podemos decir que nuestra base de datos es ***mutable***, es decir, puede sufrir modificaciones en su estructura, desde cuando se empezó el proyecto se puede afirmar que la base de datos ha ido mutando ya que en un principio se había especificado en base a unas necesidades concretas, pero conforme el proyecto iba avanzando e implementado funcionalidades había carencias que suplir en como guardar ú obtener la información o simplemente añadir funcionalidad nueva.

A continuación, vamos a mostrar el diagrama de como se encuentra actualmente nuestra base de datos a día de hoy, pero ya decimos que no es su estructura final ya que seguramente podría variar sufrir algunas modificaciones en cuanto a relaciones o añadir tablas nuevas para añadir nueva funcionalidad.

Al ser un proyecto en desarrollo es difícil especificar una estructura clara y cerrada desde el principio, por eso nuestra base de datos cambio en base a nuestras necesidades.

En el siguiente apartado vamos a mostrar el diagrama entidad relación de nuestra base de datos.

5.2.1 Diagrama de base de datos

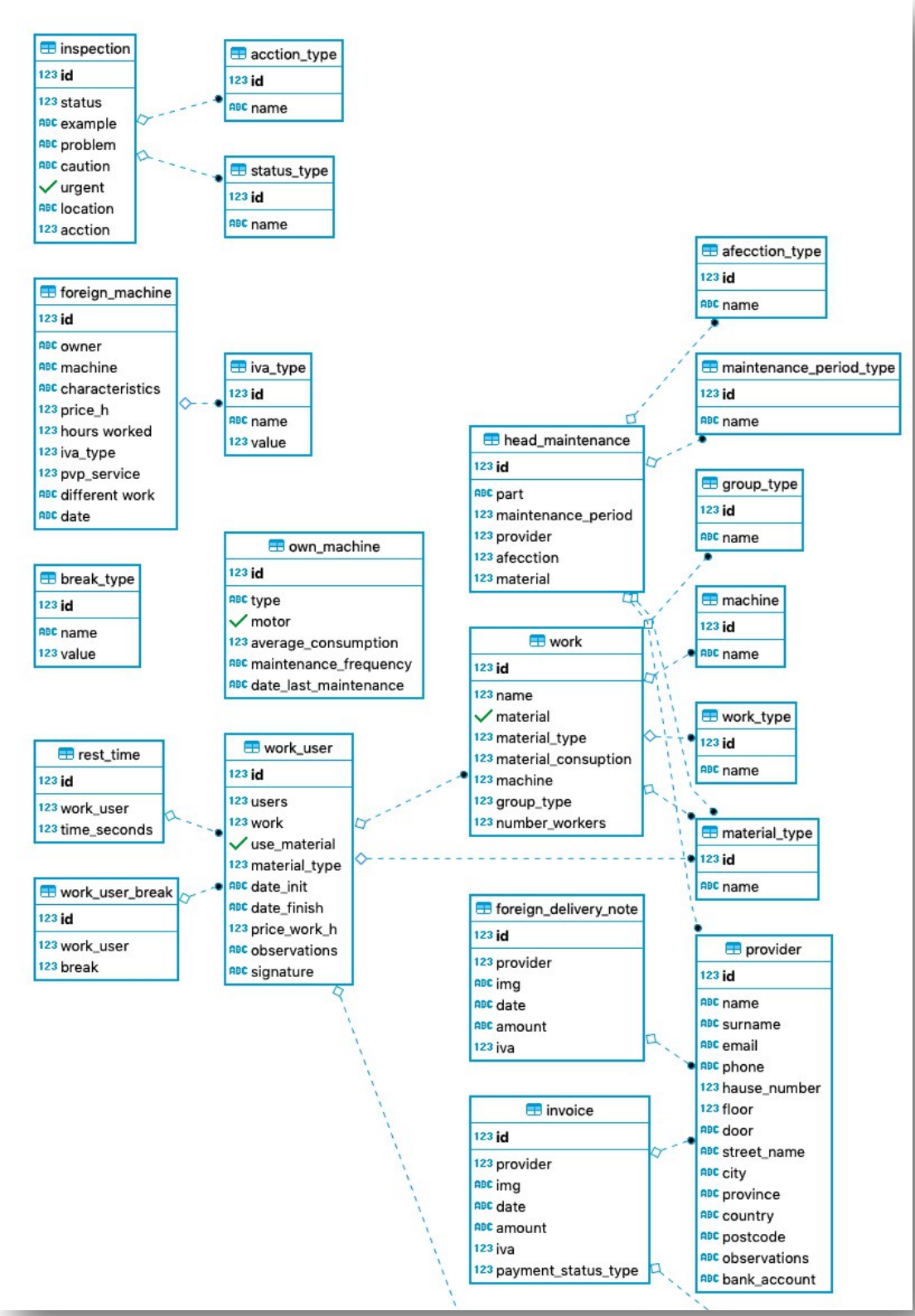


Ilustración 3. Diagrama base de datos 1

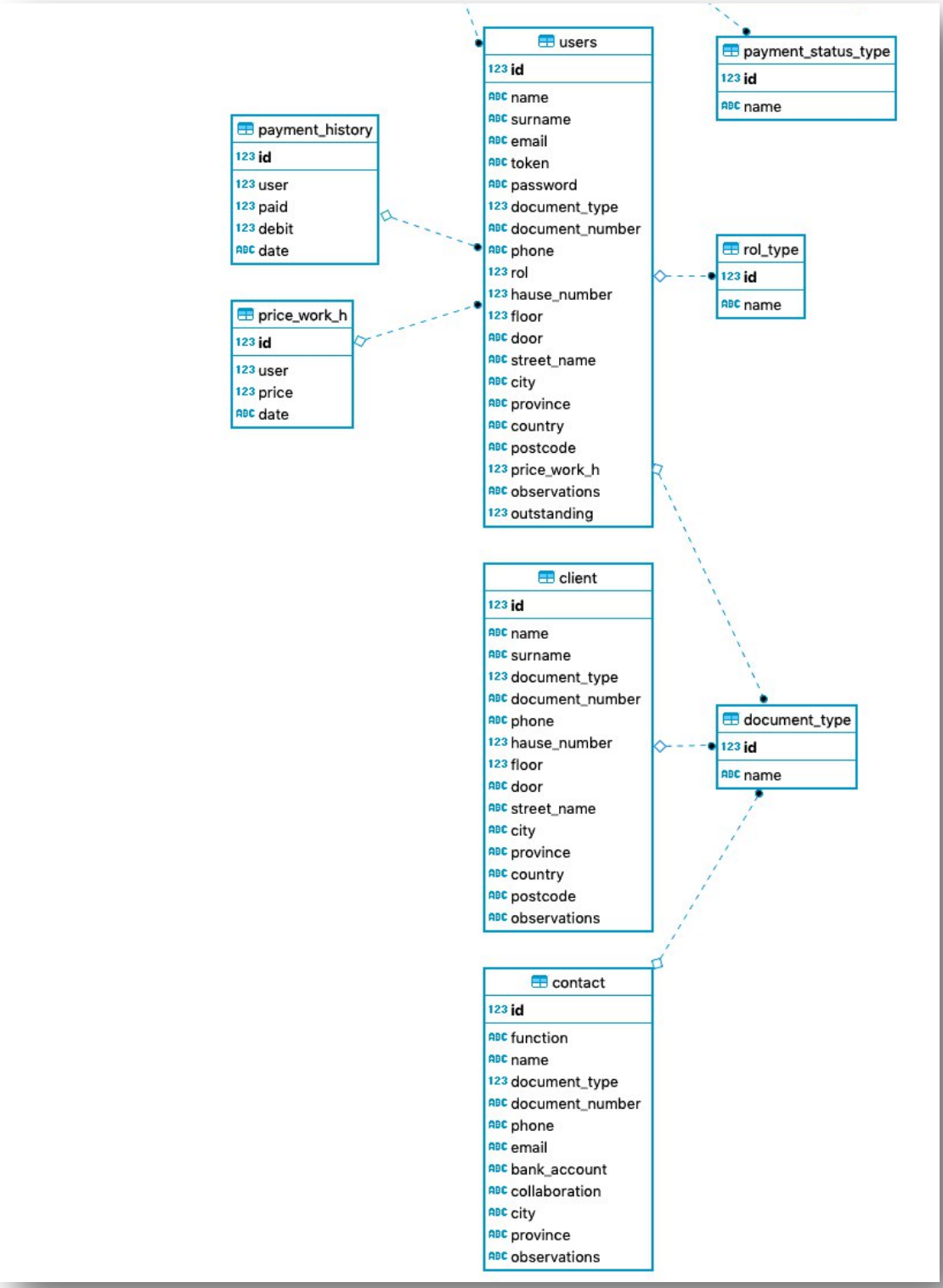


Ilustración 4. Diagrama base de datos 2

5.3 API

Como bien hemos mencionado, express.js no facilita un router de las urls. En nuestro proyecto se han definido varios métodos de ruta para poder obtener o modificar la información en la base de datos. Los usados en nuestra aplicación se definen

- **GET:** nos devuelve todo el listado como respuesta.
- **GET dado un id:** nos devuelve un único elemento como respuesta.
- **POST:** permite la inserción de un nuevo elemento.
- **PUT:** permite actualizar un elemento concreto.
- **DELETE:** permite eliminar un elemento.

Ahora bien, vamos a exponer un ejemplo de uso en nuestra aplicación. Vamos a explicar *las facturas* para ello hay definidos varios métodos donde podemos recuperar, actualizar o eliminar información.

- `GET -> GetAllInvoices = https://safe-refuge-74613.herokuapp.com/api/invoice`


```
{
    "status": "success",
    "data": [
        {
            "id": 11,
            "provider": 6,
            "img": "http://res.cloudinary.com/hidegmm1p/image/upload/files/factura1.pdf",
            "date": "1564178400000",
            "amount": "12",
            "iva": 12,
            "payment_status_type": 2
        },
        {
            "id": 12,
            "provider": 6,
            "img": "http://res.cloudinary.com/hidegmm1p/image/upload/files/factura2.pdf",
            "date": "1564178400000",
            "amount": "12",
            "iva": 12,
            "payment_status_type": 2
        }
    ],
    "message": "Retrieved ALL invoice"
}
```


Ilustración 5. Respuesta del método 'getAllInvoices', devuelve todas las facturas

- **GET** -> getInvoices = https://safe-refuge-74613.herokuapp.com/api/invoice/1

Ilustración 6. Respuesta del método 'getInvoices' devuelve una factura concreta dado un id'

- **POST** -> createInvoices = https://safe-refuge-74613.herokuapp.com/api/invoice

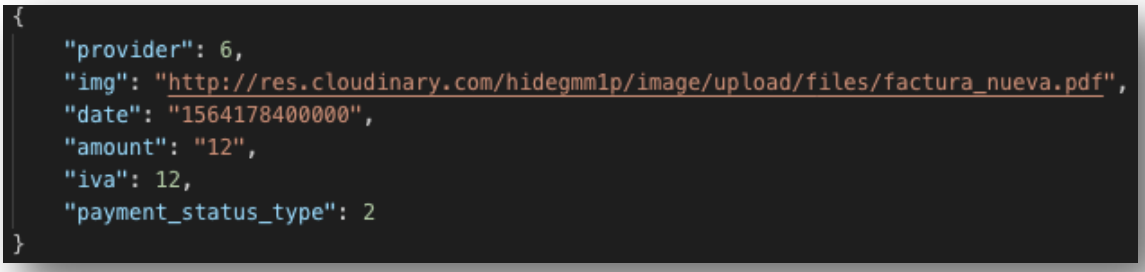


Ilustración 7. Datos de entrada para la creación de una nueva factura

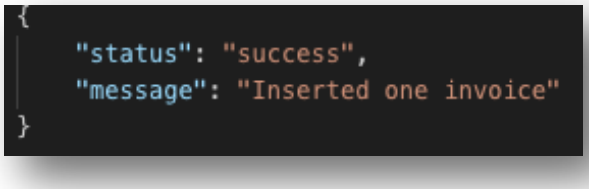


Ilustración 8. Respuesta de que un elemento se ha creado correctamente

- **PUT** -> updateInvoices = https://safe-refuge-74613.herokuapp.com/api/invoice/1

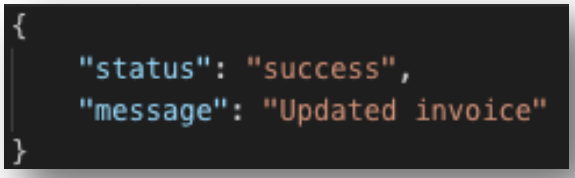


Ilustración 9. Respuesta de que un elemento se ha actualizado correctamente

- **DELETE** -> removeInvoices = https://safe-refuge-74613.herokuapp.com/api/invoice/1

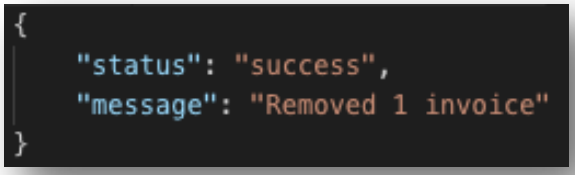


Ilustración 10. Respuesta de que un elemento se ha eliminado correctamente

5.4 Front-end web y móvil

Como complemento a la aplicación móvil desarrollada en Android se ha implementado conjuntamente también una aplicación web, donde el usuario tendrá una funcionalidad alternativa y complementaria a la aplicación móvil. La idea principal es que algunas tareas realizada desde el móvil se vea reflejado en la web y viceversa.

Desde la web se pueden realizar tareas concretas que creemos que son más tareas de escritorio por su sencillez o complejidad más que desde una aplicación móvil.

La web ofrece una serie de funcionalidades, a continuación, vamos a detallar las mismas:

- Gestión de personal: alta de nuevos empleados
- Gestión de proveedores: alta de proveedores.
- Gestión de maquinaria propia
- Inspección de viveros
- Mantenimiento
- Creación de trabajos

Por ejemplo, solo desde la aplicación móvil se podrán realizar ciertas funcionalidades como:

- Creación de tareas de trabajo y asignarlas a un empleado. También se valida mediante la firma que una tarea ha sido terminada y realiza por un cierto empleado en concreto
- Crear facturas
- Crear albaranes
- Gestión de pagos al personal e histórico de los mismos

Desde la web podremos ver estas funcionalidades a modo de consulta, pero no realizar ningún cambio sobre las mismas, nos preguntaremos porque no se pueden realizar cambios o crear estas tareas concretas. El usuario de nuestra

aplicación pasa mayor parte del tiempo fuera de la oficina, es decir, la mayor parte del tiempo la pasa en el campo o fuera. Por eso recae sobre la aplicación móvil la implementación de estas tareas ya que será accesible siempre.

A continuación, vamos a mostrar una serie de capturas de la aplicación web:

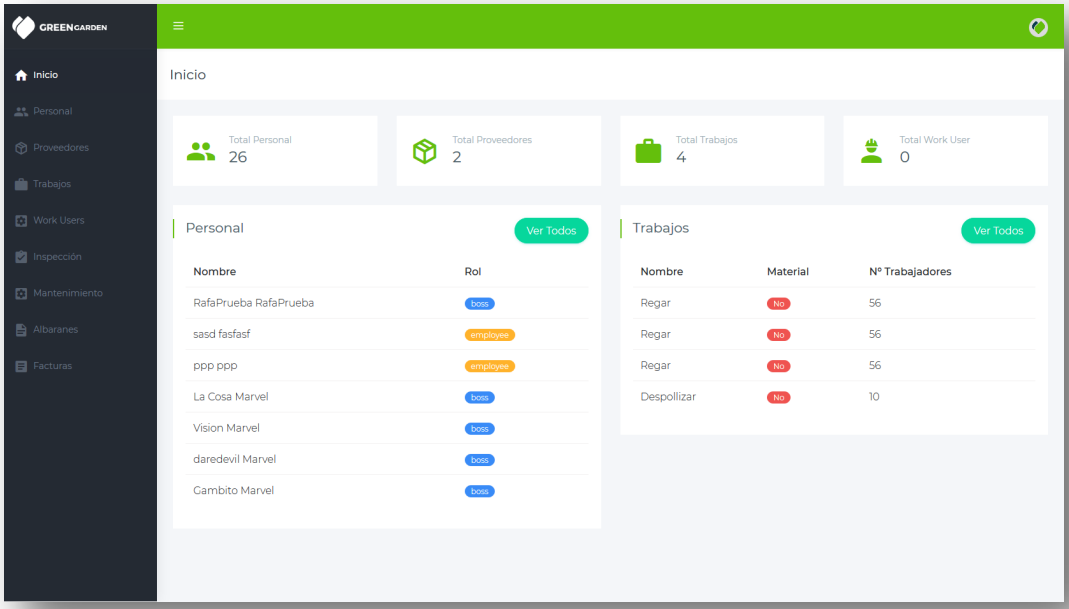


Ilustración 12. Dashboard Web

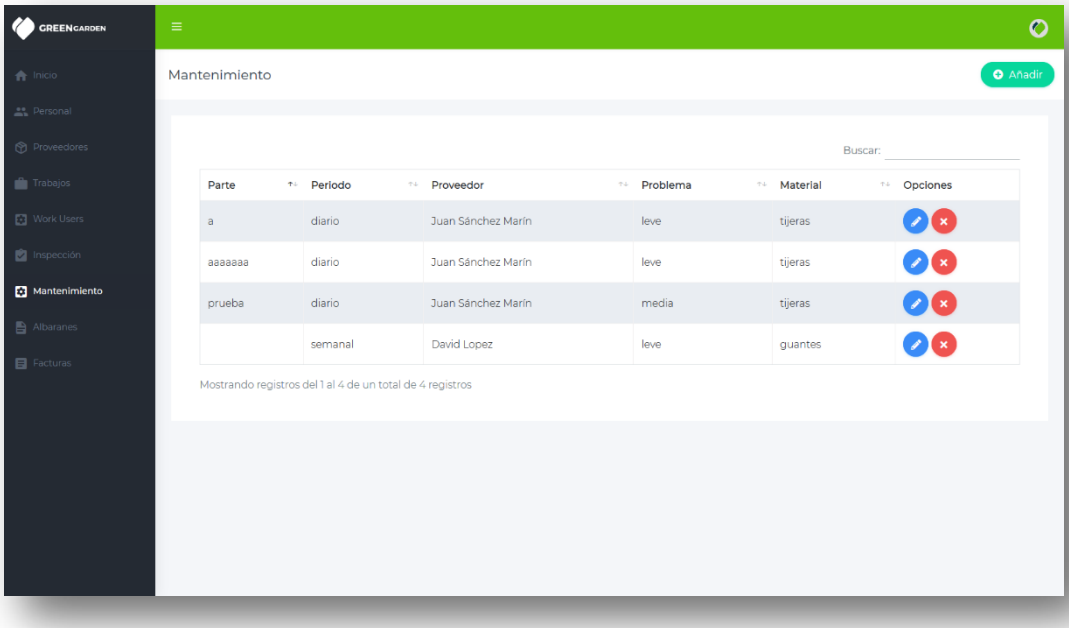


Ilustración 11. Vista de mantenimiento

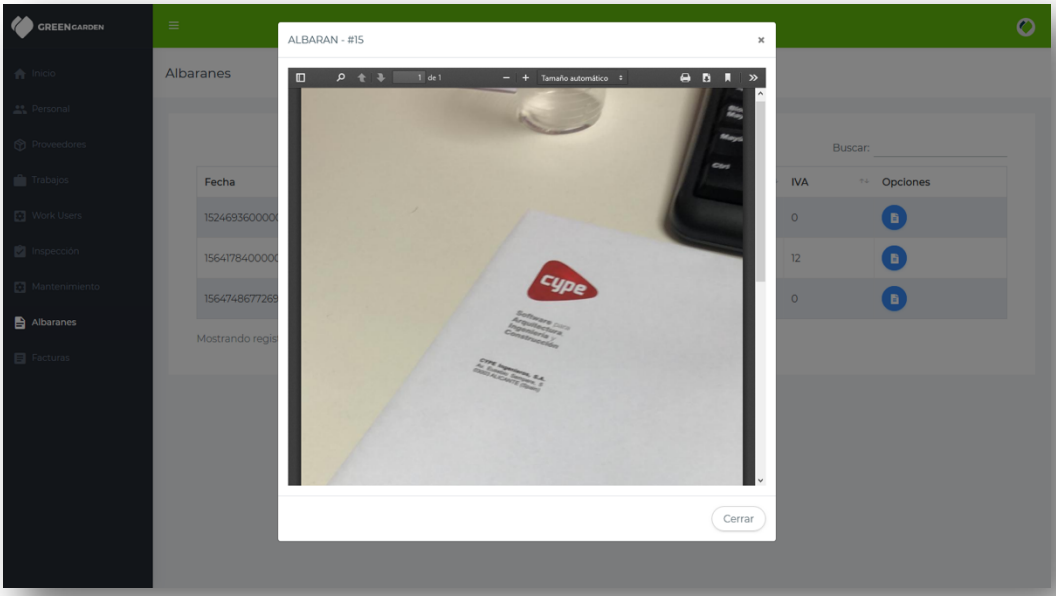


Ilustración 14. Vista de una inspección

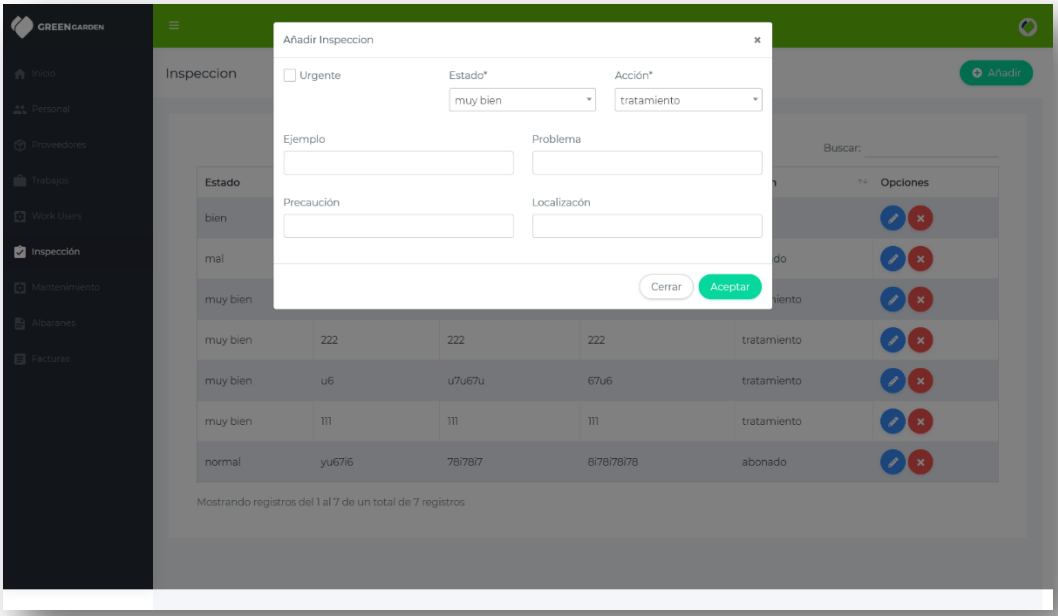


Ilustración 13. Archivo pdf creado desde la aplicación móvil

6 Implementación

6.1 Introducción

Vamos a explicar el procedimiento que se ha realizado para el desarrollo de la aplicación, explicaremos de forma detallada los pasos realizados para el desarrollo de esta y los problemas que se han encontrado durante el desarrollo. El objetivo es que veamos desde cero como se ha desarrollado la aplicación.

Explicaremos también las pantallas y cuál es su funcionalidad, así mismo se adjuntarán diversas capturas de pantalla para dar a conocer la interfaz y poder tener una idea global del funcionamiento y la interacción que tendrá el usuario con la aplicación

6.2 Base de datos basado en Room

Como hemos podido ver nuestra aplicación dispone de su propia base de datos basada en postgreSQL. Como queremos que la aplicación sea offline pues debemos implementar nuestra propia base de datos en la aplicación.

Bien como solo se implementan ciertas funcionalidades en la aplicación final ya que en la aplicación web será un complemento, ampliando cierta funcionalidad. Es decir, creemos que es más pertinente realizar algunas acciones desde una web que desde una aplicación móvil.

Puesto que no es necesario que nos guardemos toda la información de la base de datos remota, por lo consiguiente crearemos solo las entidades y relaciones que usaremos en nuestra aplicación.

A continuación, podemos observar un esquema del flujo de interacción entre las actividades/fragments y la base de datos. La ventaja de usar este nuevo paradigma es que cualquier cambio realizado en la base de datos se verán reflejado en la vista, para ello hay que definir un **ViewModel** con un **LiveData**. Con esto nos garantizamos si por ejemplo si hay una sincronización de datos y la base de datos se ve modificada, de forma automática veremos reflejado este cambio en nuestra vista.

A continuación, se muestra el esquema como funciona **RoomDatabase:**

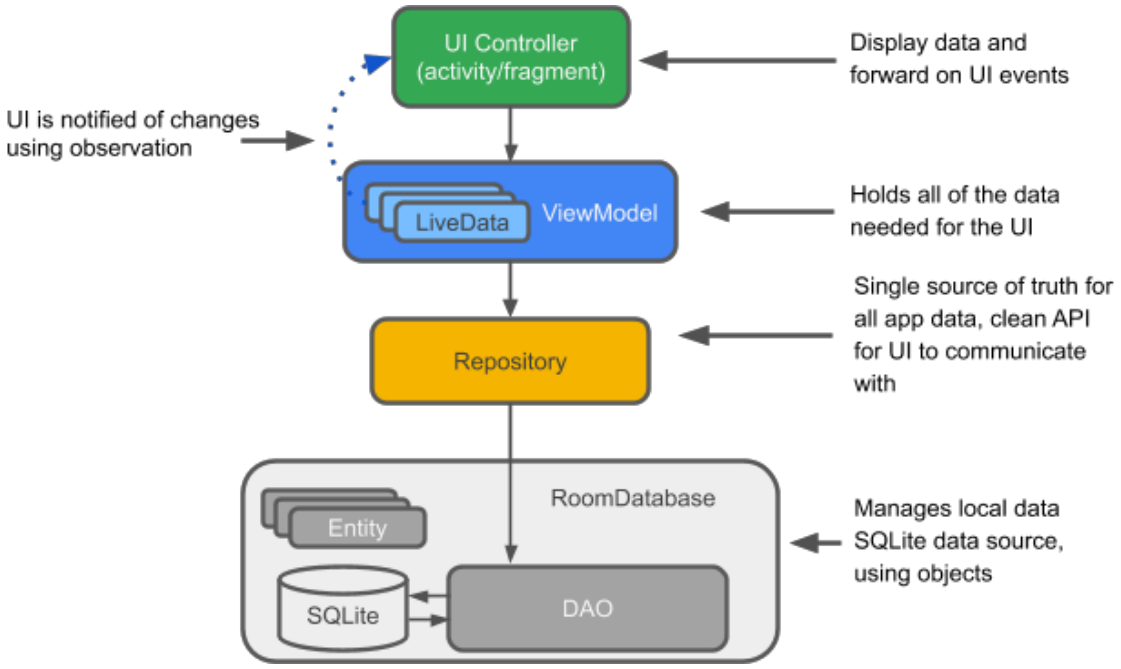


Ilustración 15. Room database

6.2.1 Entidades

Como bien hemos dicho anteriormente, en la aplicación móvil solo se implementarán ciertas funcionalidades debido a que en la aplicación web es una aplicación y hay funcionalidades adicionales.

Las tablas en **Room** se definen como entidades, son objetos que hay que definirlo con una serie de etiquetas, en el siguiente ejemplo podemos verlo:

```
@Entity(tableName = "iva_type")
public class IvaType
{
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "_id")
    private int id;

    @ColumnInfo(name = "idServer")
    private String idServer;

    @ColumnInfo(name = "name")
    private String name;

    @ColumnInfo(name = "value")
    private String value;
}
```

Se ha definido la tabla del IVA que cuenta con una id, un nombre y un valor.

6.2.2 DAO

El ***DAO*** es una interfaz que especifica los métodos con los que accederemos a la entidad en la base de datos. En nuestro caso vamos a realizar varias operaciones como:

- Insertar, actualizar, eliminar, listar elementos.
- Devolver un elemento dado una serie de parámetros.

Una cosa que podemos observar es en que el método *getAllIvaTypes*() devuelve un objeto de tipo LiveData. Este objeto es el que estará enganchado a la vista y la ventaja es que cualquier cambio realizado en la base de datos se vera reflejado de forma instantánea en nuestra vista o donde este enganchado,

```
@Dao
public interface IvaTypeDao extends BaseDao<IvaType>
{
    @Query("SELECT * FROM iva_type where _id = :id")
    IvaType getIvaTypeById (long id);

    @Query("SELECT * FROM iva_type where idServer = :idServer")
    IvaType getIvaTypeByIdServer (String idServer);

    @Query("SELECT * FROM iva_type ORDER BY value DESC")
    List<IvaType> getListIvaType ();

    @Query("SELECT * FROM iva_type ORDER BY value DESC")
    LiveData<List<IvaType>> getAllIvaTypes ();

}
```

6.2.3 Base de datos

La creación de la base de datos es tan simple como crear una clase y que extienda de *RoomDatabase*. Como podemos observar todas las entidades se deben estar definidas en la variable ***entities*** = {*lista de entidades*}.

```
@Database(
        entities = {BreakType.class, DeliveryNote.class, DocumentType.class,
GroupType.class, Invoice.class, Machine.class, MaterialType.class,
PaymentStatusType.class, Provider.class, RolType.class,User.class, Work.class,
WorkType.class, WorkUser.class, WorkUserMaterialType.class,
PaymentHistory.class, IvaType.class, WorkUserClasification.class,
WorkUserBreak.class},
        version = version_db,
        exportSchema = false)
@TypeConverters({Converters.class})
public abstract class SafeRefugeDBInstance extends RoomDatabase
{
    . . .
}
```

6.3 Sincronización

En este apartado vamos a explicar cómo se realizar la sincronización de los datos con la base de datos local de nuestra aplicación

Se han definido varios métodos de sincronización, el primero tiene lugar cuando iniciamos la aplicación por primera vez, es decir, cuando hacemos login en la aplicación y es correcto, obtenemos una serie de datos como:

- **Datos básicos:** tipos de materiales, tipos de IVA, tipos de trabajo, tipos de maquinaria, tipos de descanso
- **Datos complejos:** usuarios, proveedores, trabajos de usuario, albaranes, facturas, descansos asignados, etc.

Toda esta información se almacenará en nuestra base de datos local, de forma que podemos acceder a esta información incluso sin conexión.

Otra forma de mantener actualizada nuestra aplicación es después de realizar acciones como crear facturas, justo después de que el proceso de creación haya sido correcto, pues hacer una llamada al servidor para actualizar solo aquellos datos de la vista donde se encuentre el usuario.

Mediante notificaciones push que el servidor enviara a la aplicación con el fin de actualizar información o realizar acciones.

6.3.1 ¿Cómo mantenemos los datos actualizados con respecto al servidor?

Cada vez que entremos en la aplicación se realizar un proceso de sincronización de datos, pero esta vez no actualizamos todos los datos de nuestra base de datos local, sino solo aquellas que hayan sufrido cambios, el servidor es las respuestas nos devuelve un variable que es *dataLastChange*, con este parámetro lo que comprobamos en nuestra base de datos es si la fecha del último cambio es mayor a la que tenemos en nuestra base datos pues actualizamos los valores, en caso contrario entendemos que la información esta actualizada De esta forma nos aseguramos que tenemos los datos de nuestra aplicación siempre actualizada.

Otra forma de mantener la información de nuestra base de datos actualizada es que cada vez que realizamos una operación ya sea por ejemplo

crear una nueva factura, crear un nuevo trabajo de usuario, realizar un pago entre otras muchas tareas, después de realizar la acción actualizamos pues solo los datos concretos, es decir, si creamos una nueva factura de usuario, pues actualizaremos solo el listado de facturas y no todas las listas.

6.4 Aplicación móvil

En este apartado vamos a desarrollar y explicar el funcionamiento de la aplicación, así mismo se explicarán las distintas funcionalidades de esta y además adjuntaremos distintas capturas de pantalla de la aplicación. Vamos a explicarlo en diferentes apartados. Se explicará la aplicación principalmente desde el punto de vista con un rol de tipo administrativo, ya que es donde se puede aprovechar al máximo las funcionalidades implementadas.

6.4.1 Componentes implementados

Durante el desarrollo de la aplicación se realizo un análisis previo de todos los componentes, fragments, actividades y layouts que conformarían la aplicación.

Un objetivo importante es que la aplicación a nivel visual presente homogeneidad, para esto debemos definir varias cosas:

- Estilos para los componentes que serán compartidos.
- Colores que definirán toda la aplicación
- Clases abstractas con comportamiento común
- Actividades abstractas
- Diálogos abstractos, que solo cambiaran su contenido. Pero los botones, títulos y demás cosas tendrán un nivel visual común

Lo que se pretende es tener una aplicación robusta, fácil de desarrollar y sea sencilla de encontrar errores y a nivel visual que presente homogeneidad y ante cualquier posible cambio o imprevisto pueda adaptarse. Para este fin se ha intentado abstraer el máximo numero de clases, layouts, funciones similares. De estar forma no evitamos repetir código en muchas clases ya que será similar, y nos permitirá centrarnos en lo verdaderamente importante que es la implementación de la aplicación

6.4.1.1 Diálogos

En los diálogos, se ha extraído a nivel visual unos componentes comunes y iguales que se repetirán en todos los diálogos:

- Icono junto al titulo del dialogo
- Titulo
- Botones de aceptar y cancelar la acción
- Un *linearlayout* para el contenido del dialogo

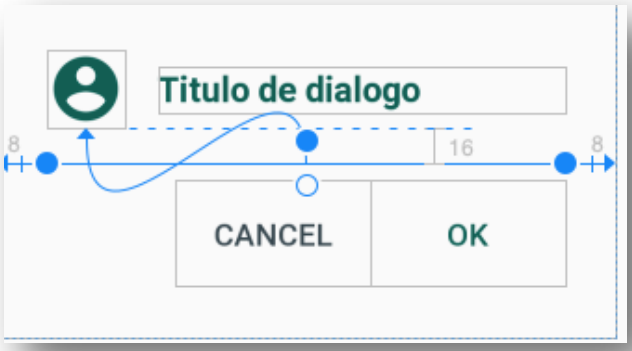


Ilustración 16. Díalogo abstracto vista

La clase *AbstractDialogFragment* define un comportamiento común, también ofrece funciones abstractas que serán implementadas en las que hereden de ésta, estas funciones son:

- *protected abstract boolean* ***onCanAccept*** *();*
- *protected abstract void* ***onClickAccepted*** *(Intent intent);*
- *protected abstract View* ***getView*** *(LayoutInflater inflater);*
- *protected abstract String* ***getTitleDialog*** *();*
- *protected abstract String* ***getTextButtonSubmitDialog*** *();*

6.4.1.2 Fragments con listas

En las listas hay un patrón que se repite y está presente en varias vistas de la aplicación es el listado de facturas, albaranes, usuarios, etc. Para esto se creo una abstracción de un *fragment* compuesto por:

- Un *recyclerview* como elemento principal donde se mostrarán las vistas, cada vista implementará una función *getAdapter ()* donde se devolverá un *adapter* personalizado que se asignará al *recyclerview*.

- Un *floating action button* que se mostrará con el símbolo de +, esto se define como un componente asociado a una función abstracta y cuyo comportamiento se definirá en cada vista.

Ilustración 17. Fragment abstracto vista

Las funciones definidas en la clase *AbstractBaseListFragment* son las siguientes:

- *protected abstract String* ***getTitleActionBarIfChange*** *();*

- *protected abstract AbstractRecyclerViewAdapter* ***getAdapter*** *();*

- *protected abstract IViewModel<T>* ***getViewModel*** *();*

- *protected abstract void* ***onCreateViewSynchronizeDataFromServer*** *(final String authToken)*

- Esta acción esta asociada a un ***SwipeRefreshLayout*** que nos permitirá actualizar los datos de la vista llamando al servidor.

- *protected abstract void* ***buttonAddClick*** *(View view);*

Otra cosa que común a todas las vistas es cuando no hay datos, se muestra un mensaje con el siguiente contenido:

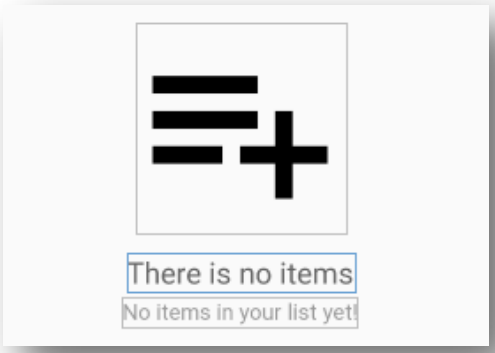


Ilustración 18. Fragment, vista de no hay elemento.

Para esto también se han definido unas funciones abstractas:

- protected abstract int **getTitleNoItemFound** ();
- protected abstract String **getDescriptionNoItemFound** ();

En nuestra vista se podría ver de la siguiente forma:

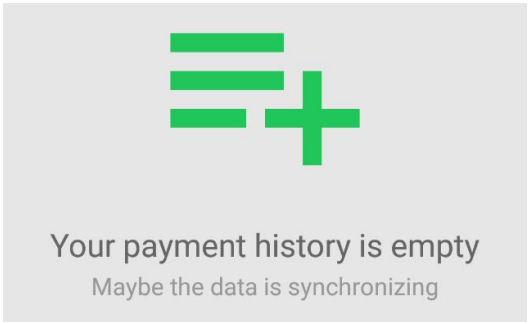


Ilustración 19. No hay elementos en el historial de pagos.

6.4.2 Aplicación con rol usuario

La aplicación desde el punto de vista con rol usuario, es básica ya que cuenta con solo 2 funcionalidades a modo consulta, es decir, el usuario no podrá realizar ninguna acción:

- **Listado de trabajos:** podemos ver un listado de los trabajos pendientes o realizados, no podemos realizar ninguna acción sobre ellos, ya que es a modo consulta. El administrador es quien puede realizar acciones sobre los trabajos.

- **Historial de pagos:** Se puede tener un histórico de todos los pagos realizados a un usuario concreto, es igual a modo consulta ya que no se puede realizar ninguna acción sobre ellos

- **Perfil de usuario:** Una pantalla simple donde mostramos información básica del usuario.

A continuación, vamos a mostrar unas capturas simples de la aplicación desde el punto de vista usuario:

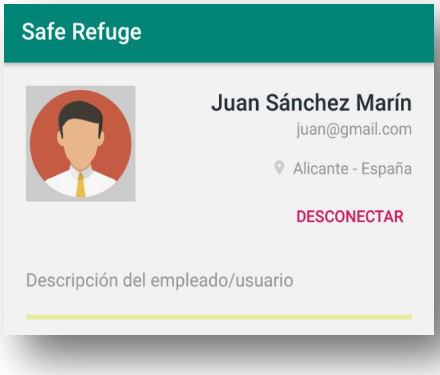


Ilustración 20. Vista perfil de usuario

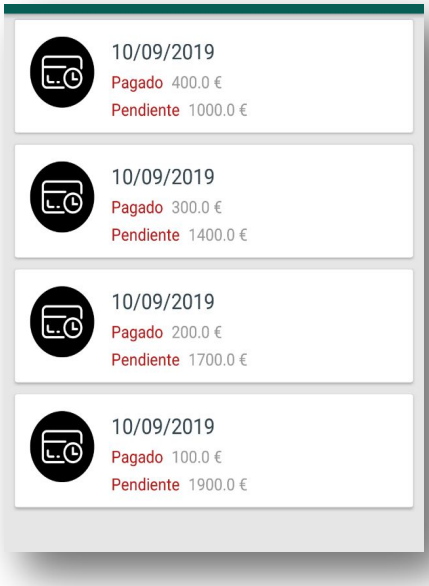


Ilustración 21. Listado de historial de pagos

6.4.3 Login

Este es el punto de entrada a la aplicación, en este apartado se difiere el rol que el usuario tendrá cuando se realice el login. Como mencionamos previamente existen 2 roles, el rol usuario ya explicado en el apartado anterior y el rol de administración que es el que explicaremos en profundidad a lo largo de los siguientes apartados

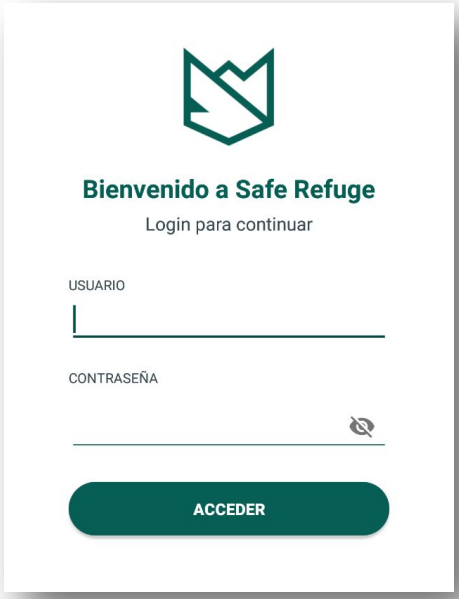


Ilustración 22. Pantalla de login

6.4.4 Albaranes

Desde la aplicación móvil se podrán crear albaranes con unos datos específicos, y además hay un listado con un histórico de los mismos. Esta funcionalidad es exclusiva de la aplicación móvil, por otro lado, desde la aplicación web se podrán acceder a ellos solo a modo de consulta en ningún lado se podrán realizar acciones sobre ellos.

Vamos a mostrar como es el proceso de creación albaranes desde la aplicación móvil. En la siguiente ilustración tenemos la vista que el administrador podrá usar para crear un albarán

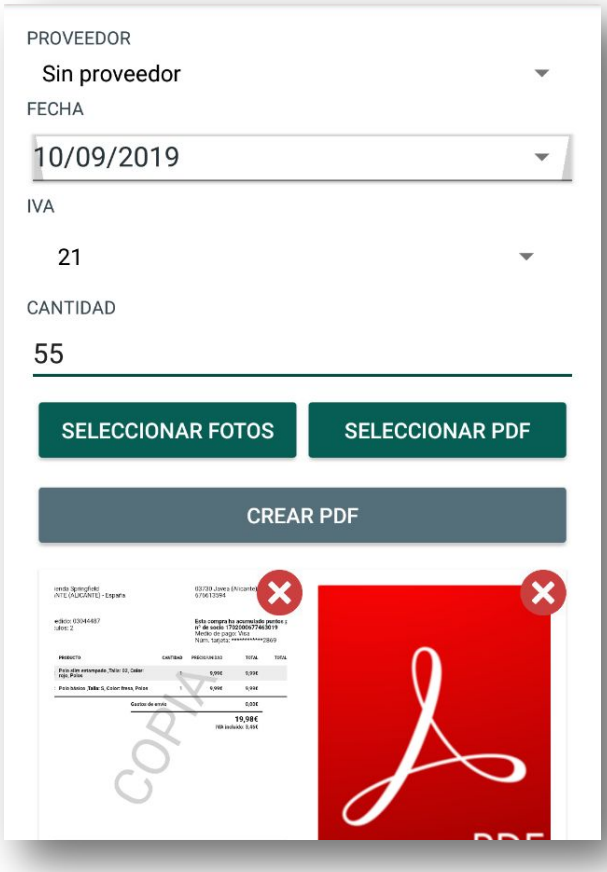


Ilustración 23. Crear un albarán

Como podemos ver hay varios componentes en la vista:

- **Spinner proveedor:** aquí mostramos una lista donde el usuario podrá seleccionar un proveedor.
- **Botón para seleccionar una fecha:** por defecto se marca la fecha de día de hoy, pero el usuario podrá modificar la fecha haciendo clic en el botón, al hacer clic se abrirá un dialogo con un calendario donde podrá seleccionar la fecha que él crea adecuada
- **IVA:** otro desplegable con varias opciones de IVA, en caso de que el usuario no le apañe ninguna de las opciones, podrá escribir su propio IVA, en las opciones del despegable hay una opción de IVA personalizado
- **Cantidad:** la cantidad total del albarán
- **Botones:** estos botones tienen varias opciones y funcionalidades

- **Seleccionar fotos:** podemos acceder a las fotos de nuestra galería o bien realizar una foto
- **Seleccionar pdf:** podemos acceder a los documentos de nuestro móvil
- **Crear pdf:** el usuario podrá crear y previsualizar el archivo en formato pdf si quisiera.

Una vez creado un albarán se puede ver en el listado (también podrán consultarse desde la web) y acceder a el, con el fin de ver los datos y ver el documento adjunto, así mismo se permite la edición de un albarán por si se quisiera realizar alguna modificación sobre el mismo. A continuación, mostramos unas fotos de las dos vistas

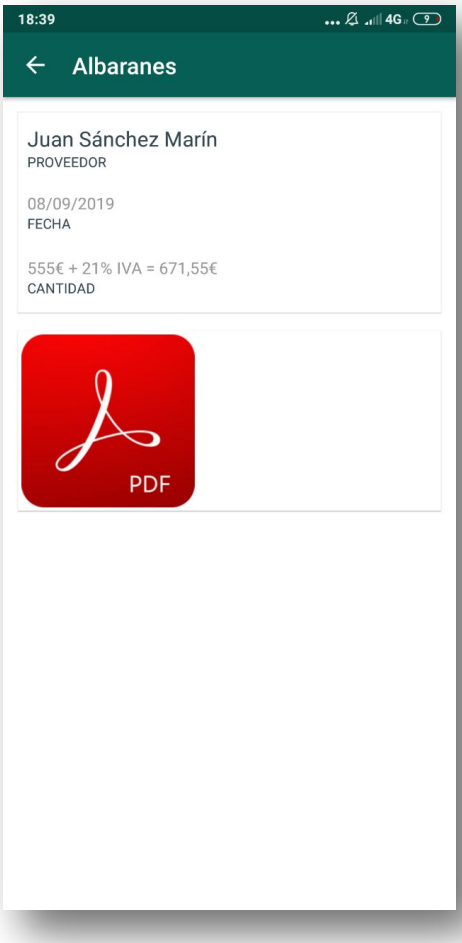


Ilustración 24. Vista detalle de un albarán

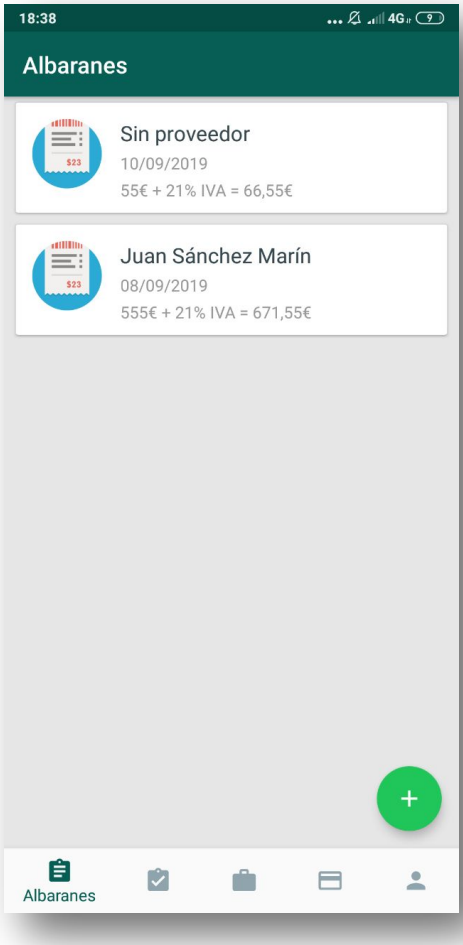


Ilustración 25. Listado de albaranes

6.4.5 Facturas

El funcionamiento de las facturas es similar al de los albaranes, tenemos una serie de opciones que el usuario puede elegir, una vista detalla para consultar información de una factura concreta, pero lo que difiere es el estado en el que puede estar:

- **Pendiente:** una factura puede estar en estado pendiente si no se ha pagado todavía al proveedor. Una vez se haga un pago se podrá marcar como pagada y pasara a estado pagada. Se puede diferenciar porque tiene un tick en color rojo
- **Pagada:** una factura se encontrará en estado pagada, podremos diferenciarlas porque tienen un tick en color verde

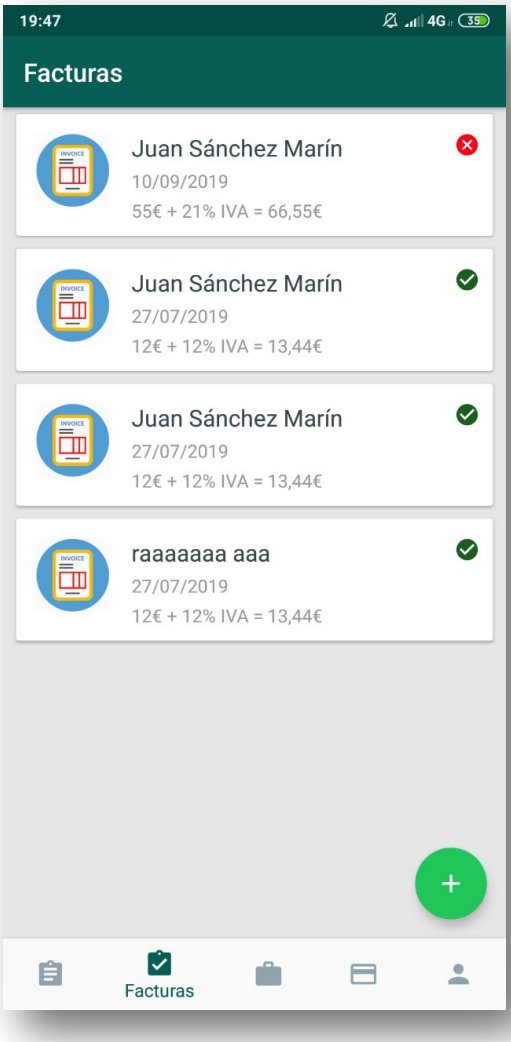


Ilustración 27. Listado de facturas

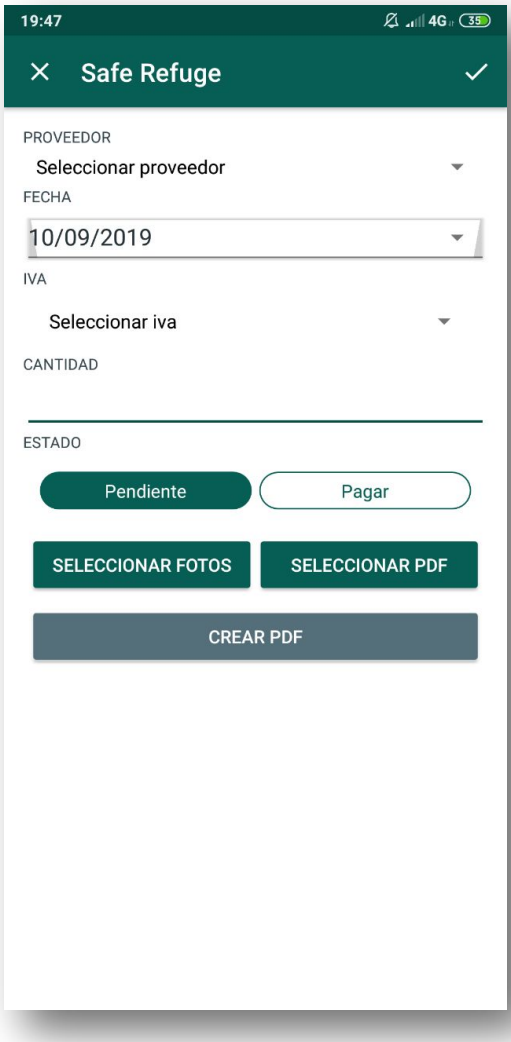


Ilustración 26. Crear una factura

6.4.6 Pagos

En esta vista el usuario con el rol administrador dispone de un listado de todos los usuarios con cierta información. En la vista esta compuesta por el nombre del usuario y la cantidad pendiente de pago.

A cada empleado se le podrán realizar pagos en esta vista, el modo de pago es él elegido por el empleado, puede ser un abono en su cuenta bancaria o bien un pago en efectivo.

Además, por cada usuario se dispone del histórico de pagos realizados de cada uno, la diferencia es que en este caso tenemos acceso al histórico de todos los usuarios.

A continuación, vamos a mostrar unas capturas de esta funcionalidad:

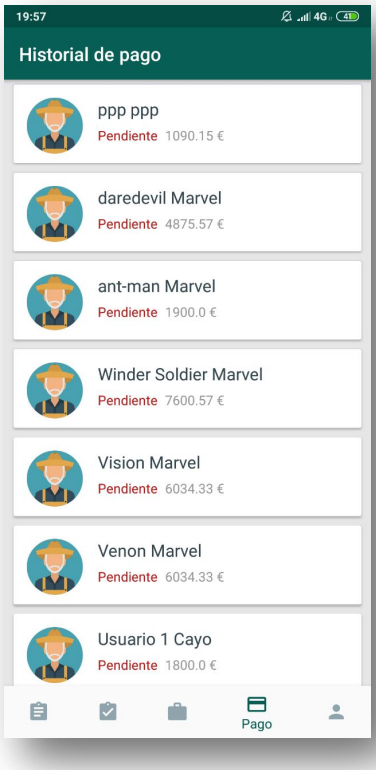


Ilustración 29. Listado de usuarios

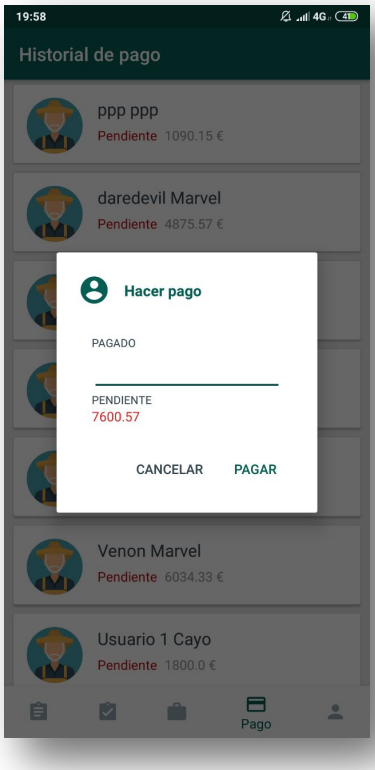


Ilustración 30. Realizar un pago

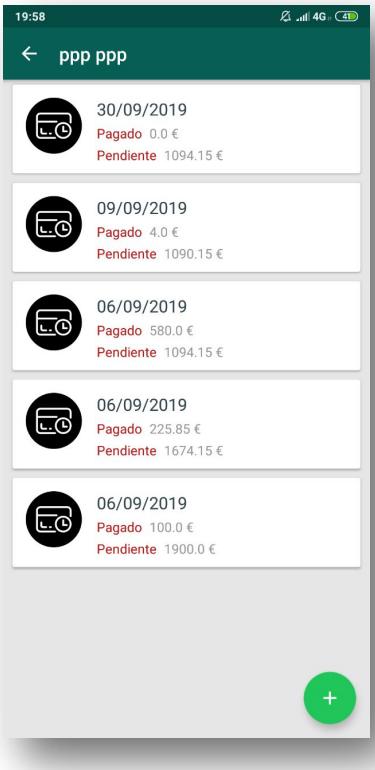


Ilustración 28. Histórico de pagos realizados de un usuario concreto

6.4.7 Trabajos de usuario

Este apartado es la parte importante de la aplicación, ya que desde aquí se gestiona los distintos trabajos realizados por los usuarios. También se pueden controlar diferentes aspectos:

- los descansos de cada trabajador,
- los materiales usados para realizar cada trabajo, de esta forma se podrá hacer un análisis del coste que tiene cada trabajo
- finalizar una tarea
- validar mediante una firma que el trabajo se ha realizado por un usuario ya, con esto podemos realizar un informe de las horas trabajadas de cada usuario.

Vamos a desarrollar en varios apartados la funcionalidad de los trabajos de usuarios.

6.4.7.1 Creación de tareas

Desde la siguiente vista podemos realizar una serie de acciones:

- Crear una tarea y asignarle un trabajo especifico.
- Asignar uno o varios usuarios, como podemos ver en la ilustración podemos seleccionar varios usuarios que realizaran la tarea.
- Podemos elegir si la tarea requiere materiales o no, es caso afirmativo podemos seleccionar múltiples materiales.
- Se puede seleccionar la fecha de inicio de la tarea, por defecto se marca la fecha y hora del día, casi siempre se tratan de tareas que tiene una duración superior 1 hora y como máximo una duración de 8 horas, lo que dura la jornada.
- Se puede predefinir la hora fin si son tareas cerradas y con una duración determinada, o podemos finalizar una tarea cuando esta se haya terminado.
- Podemos añadir notas a una tarea en el campo observaciones.

Con toda esta información podremos crear una tarea con varios usuarios asignados a ella y con ciertas características.

Ahora bien, después de la creación de la tarea, podemos verla en nuestro listado e incluso en la aplicación web a modo de consulta.

6.4.7.2 Listado de tareas

Una vez creada la tarea nos aparece en el listado de tareas, para diferenciar las tareas pendientes de las terminadas se ha añadido un tick en color verde cuando este finalizada o un tick en color rojo cuando la tarea esta en curso o pendiente de terminar.

Desde esta vista podemos realizar la acción de finalizar una tarea, cuando se quiera finalizar una tarea tenemos que seleccionar la tarea y nos aparecerá un dialogo con un calendario para seleccionar el día y spinner para seleccionar la hora concreta, por defecto están marcadas al día y hora de hoy.

A continuación, lo podemos observar en las ilustraciones lo mencionado anteriormente:

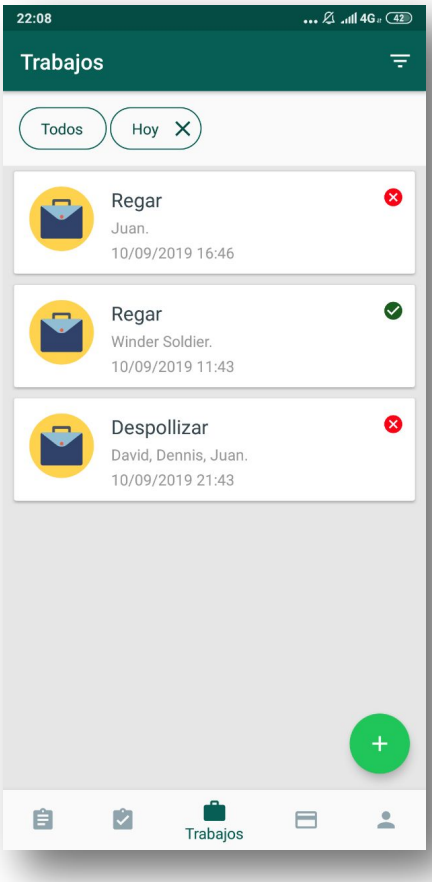


Ilustración 32. Listado de trabajos de usuario

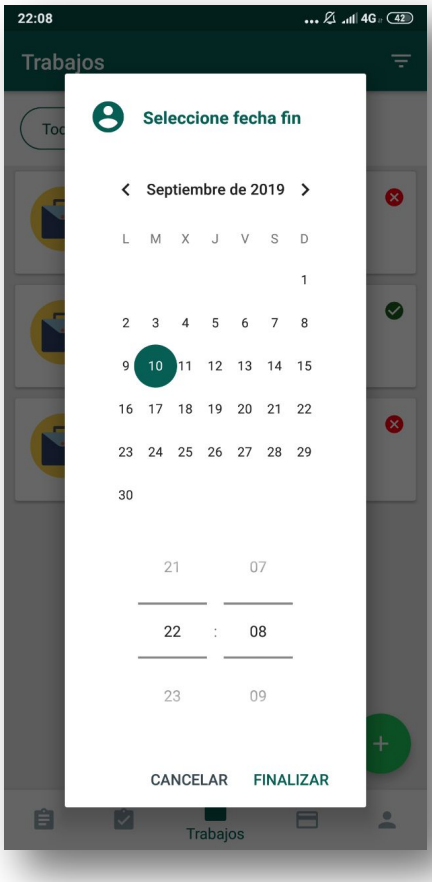


Ilustración 31. Dialogo para finalizar una tarea

Cuando se realiza la acción de finalizar una tarea, se realizan una serie de cálculos, en primer lugar, se calculan las horas trabajadas, luego se restan los descansos que ha tenido el usuario, esta cantidad de horas se multiplica el precio por hora que cobra un trabajador y por ultimo se añade la cantidad a pagar a la cantidad pendiente de pago. De esta forma conocemos en todo momento cuando se le debe abonar a un trabajador por las diferentes tareas realizadas.

6.4.7.3 Filtro

Dado que podemos tener una gran cantidad de tareas distintas por usuarios en un día o un periodo de tiempo. Se ha añadido una funcionalidad que nos permite filtrar las tareas realizadas por tareas o fechas o ambas inclusive. A continuación, vamos a mostrar un ejemplo de lo implementado.

Podemos seleccionar un trabajo concreto y una fecha, para el siguiente ejemplo hemos elegido la tarea regar y la fecha del día de hoy.

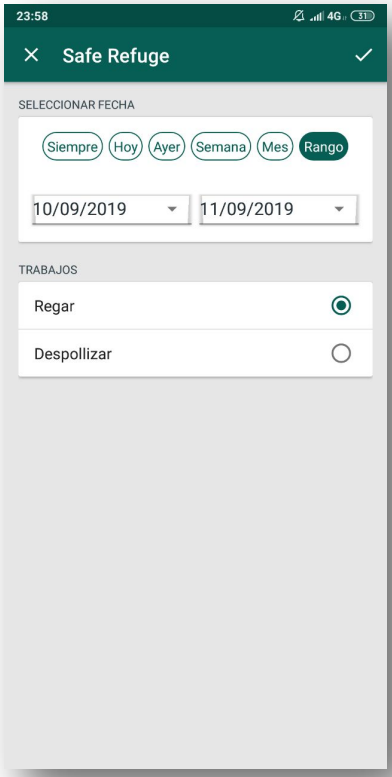


Ilustración 34. Vista de filtro de trabajos de usuario.

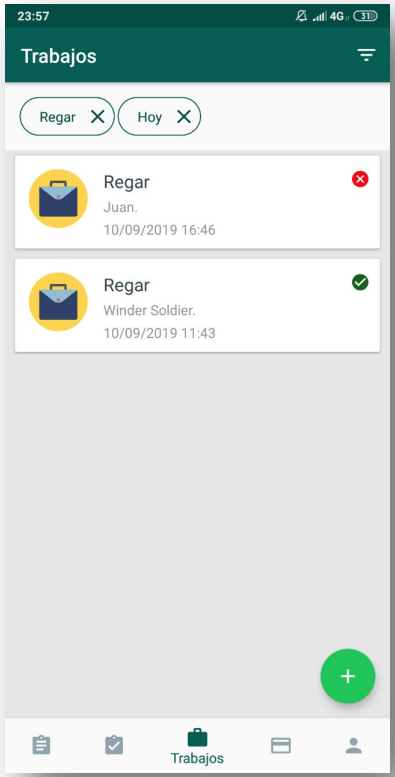


Ilustración 33. Resultado después de aplicar los filtros

Como podemos observar en la vista del filtro el usuario podrá elegir entre dos opciones:

- **Fecha:** podrá elegir entre distintas opciones:
  - **Siempre:** todo el histórico sin filtrar por fechas.
  - **Hoy:** día de hoy.
  - **Ayer.**
  - **Semana:** semana actual.
  - **Mes:** mes actual,
- **Trabajos:** aquí se mostrará un listado con todos los trabajos disponibles.

En la ilustración 31, podemos observar la lista filtrada después de haber aplicado los filtros. Podemos observar que hay dos botones en modo texto arriba con los filtros aplicados, haciendo clic sobre ellos eliminaremos el filtro aplicado y se volverá a mostrar la lista con la configuración de filtros aplicada.

6.4.7.4 Vista detalle de un trabajo

Después de crear un trabajo de usuario, podemos acceder a los detalles de cada uno a través del listado.

Una vez dentro de la vista detalles, tenemos información detallada de trabajo, como:

- Los materiales que se usan.
- La fecha de inicio y la fecha de finalización de un trabajo.
- Un campo extra llamado observaciones para que los trabajadores tengan en cuenta algunos detalles para cuando se realice un trabajo.

Además, en esta vista podemos observar un listado de los usuarios asociados a ese trabajo concreto, esto es importante porque desde aquí es donde podemos validar que un usuario o usuarios han realizado una tarea.

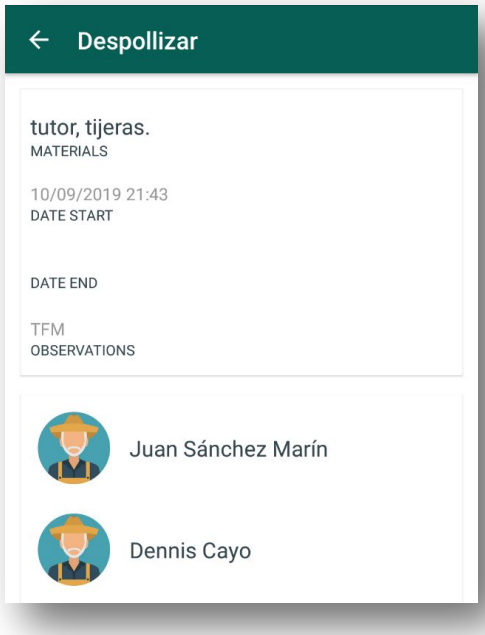


Ilustración 35. Vista de un trabajo

Si se quiere conocer más detalles del trabajador asociado a esta tarea, podemos ir a la vista de cada usuario donde encontraremos información adicional como el precio por hora que percibe el empleado por la tarea realizada

6.4.7.5 Descansos

Dentro de la vista del trabajador tenemos otra opción la de añadir descansos a un trabajo. Cada tarea puede tener asociadas una serie de descansos, que son propios de cada trabajador. Vamos a exponer un ejemplo para ver su uso.

Un trabajador empieza a realizar una tarea concreta, ante un imprevisto debe salir del trabajo y necesita ir a realizar una gestión. Pues se podrá añadir un descanso asociado a esa tarea, esto es útil porque su salario se calcula en base a las horas trabajadas.

Esta información se usará posteriormente para calcular la cantidad total que se le debe abonar por la realización del trabajo.

Para añadir un descanso se muestra un desplegable con una serie de valores predefinidos, pero si se diera el caso de que ningún valor se adecua a la necesidad, se podrá escribir en minutos el tiempo del descanso.

A continuación, mostramos un ejemplo de los descansos:

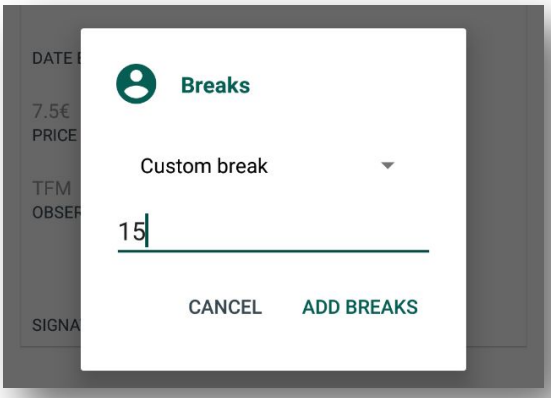


Ilustración 36. Descanso personalizado

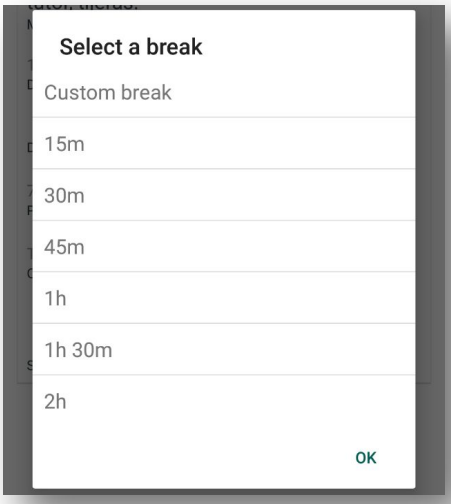


Ilustración 37. Tipos de descansos

6.4.7.6 Validar trabajo

El pasado 12 de marzo, se publicó en el BOE cómo será el registro de horario. A continuación, vamos a explicar algunos requerimientos básicos del registro de horario de los empleados:

- Todas las empresas deberán llevar el registro de horario de sus empleados independientemente de la jornada que estos tengan.
- La empresa estará obligada a guardar el registro de horario de los empleados durante cuatro años.
- Así mismo, deberá estar disponible para los empleados y los sindicatos.
- Los empleados deben conocer la distribución y duración de la jornada laboral ordinaria.
- Los sindicatos deben conocer mensualmente las horas extras realizadas por los empleados.

A partir del 12 de mayo de 2019 todas las empresas deberán registrar la jornada laboral de los empleados. Además, las empresas deben garantizar el registro horario de la jornada que deberá incluir el horario concreto de inicio y finalización de la jornada de trabajo de cada persona trabajadora, sin perjuicio de la flexibilidad horaria que establece en propio precepto estatutario.

[Referencia 37]

Como se explicado previamente, con la entrada en vigor de esta nueva ley, todas las empresas están obligadas a llevar un registro del horario de sus empleados, pues es de aquí de donde surge la idea como llevar un control del horario de los empleados y que puedan ser validados por ellos.

Para cubrir esta necesidad se ha implementado una forma de validar las horas que cada trabajador ha necesitado para realizar un trabajo concreto.

El modo de funcionamiento es el siguiente, solo cuando una tarea pase a estado finalizada se podrá acceder a esta opción, es decir, solo cuando una tarea se haya dado por finalizara aparte de realizar los cálculos oportunos de horas

trabajadas restando los descansos si se tuvieran, y añadiendo un pago pendiente al histórico de cada trabajador, solo cuando se cumplan estas condiciones en la aplicación se activará la siguiente opción.

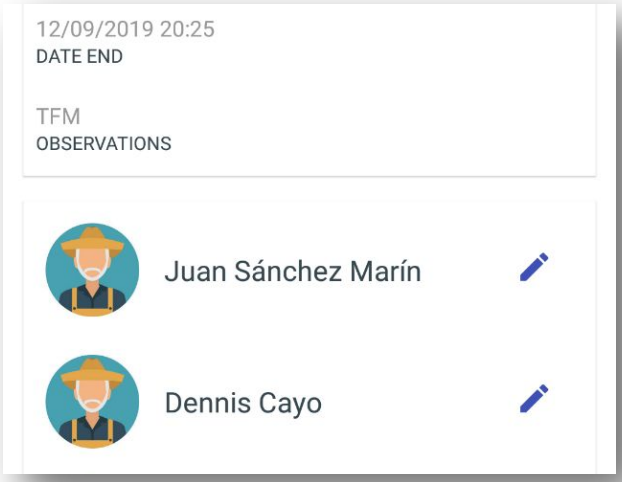


Ilustración 38. Botones para validar un trabajo

Como podemos observar se activará un botón con que brinda la opción de firmar la tarea para que sea validada por el trabajador correspondiente.

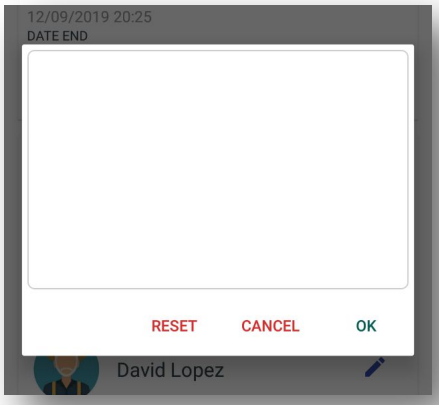


Ilustración 40. Vista para validar una tarea

Ilustración 39. Ejemplo de firma

Bien pues el trabajador después de firmar se validará la tarea y se dará por finalizada.

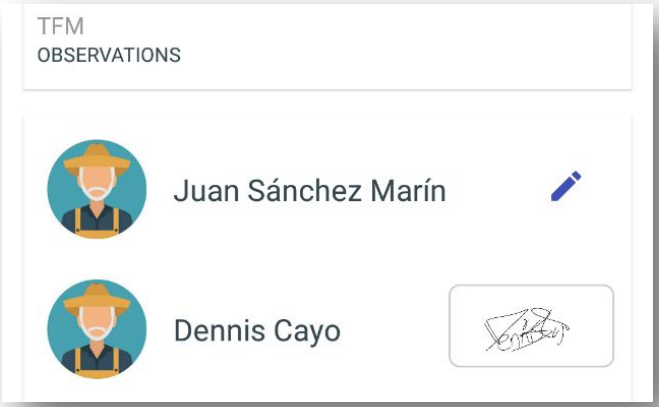


Ilustración 41. Usuario con tarea validada

Esto es una opción que puede aportar un gran valor a la aplicación, ya que se pueden generar informes estadísticos por cada trabajador. Estos informes pueden ser:

- Número de tareas realizadas en un periodo de tiempo
- Extraer el número horas trabajadas por semana para validar que se corresponda con el número de horas estipuladas en su contrato.
- Generar informes para ver el rendimiento de cada trabajador y ver donde puede ser más eficiente o en que tipo de trabajo puede desempeñarse mejor.

7 Conclusiones

Nuestra aplicación es un proyecto real que tiene como punto de partida las necesidades de una persona que acudió a varias ferias de agricultura, donde realizó demos de programas que le podrían ayudar en la automatización y gestión de tareas. Pero se llegó a la conclusión de que sus necesidades básicas en algunos programas no estaban cubiertas y en otros muchos casos eran sistemas muy complejos y difíciles de usar, o simplemente hacían más funcionalidades de la necesitada encareciendo exponencialmente el coste.

A partir de este caso individual nace nuestro proyecto que tiene el fin de cubrir estas necesidades y ayudar en la gestión y automatización de los procesos de trabajo. Hay que destacar que se especificó desde el inicio cuales eran los objetivos y el propósito del proyecto de la mano del cliente.

Una vez identificadas las necesidades, éstas pasan a ser requerimientos. Estos requerimientos se convertirán en el proyecto que se ha presentado a lo largo de este trabajo de fin de máster.

Por un lado, a lo largo del proceso de desarrollo se han hecho varias reuniones con el dueño de la aplicación para determinar si se estaban dando respuesta a sus necesidades. En la primera reunión había funcionalidades implementadas que se adaptaron a sus peticiones, y se valoro que otras tenían que ser modificadas.

A lo largo de la segunda reunión una vez realizados los cambios se estableció un nuevo requerimiento. Como consecuencia de la nueva ley:

"*Resolución de 28 de febrero de 2019, de la Secretaría de Estado de Función Pública, por la que se dictan instrucciones sobre jornada y horarios de trabajo del personal al servicio de la Administración General del Estado y sus organismos públicos.*"

[Referencia 38]

Surgió una necesidad, y por este motivo se implementó la finalización y validación de un nuevo trabajo con la firma del empleado, de esta forma se pudo realizar informes de las horas de cada empleado con el fin de garantizar el cumplimiento de la ley.

Por otro lado, durante el desarrollo de este proyecto hemos podido ver las partes por las que el desarrollo del software transcurre. Por ello, partimos de las charlas con lo clientes para captar sus necesidades y convertir estas necesidades en requerimientos, y continuamos observando cuales son las tecnologías o herramientas más adecuadas para el desarrollo del software.

Para conseguir dicho fin, se realizará un análisis de mercado y un estudio previo de la gestión agrícola con el fin de brindar al usuario una experiencia agradable y que cumpla sus expectativas usando técnicas de la ingeniería informática y el sector agrícola.

Los objetivos que hemos llevado a cabo son:

- Función multi-dispositivo para trabajar desde la aplicación web como desde el móvil.
- Todos los datos centralizados y accesibles siempre instante.
- Gestión de personal.
- Gestión de facturas y albaranes
- Gestión de trabajos de usuario.
- Control de horarios e informes.
- Estudio de mejoras de futuro como la geolocalización de maquinarias, vehículos y sistema IOT para mejorar la producción en el campo.

Actualmente el usuario tiene una demo de la aplicación web y móvil. Con ella está realizando pruebas con el objetivo de comprobar que todo funcione correctamente y además localizar posibles errores. En el caso de que las aplicaciones cumplan con sus objetivos se llevaría el sistema a producción a finales de año.

8 Trabajo de futuro

Como mencionamos previamente es un proyecto en desarrollo, pues no está implementado al 100%. Pues hay varias ideas que se implementara en un futuro.

La primera es dotar al sistema de notificaciones push, ya que mediante estas notificaciones el servidor enviara a los usuarios subscritos acciones como actualizar información, algún evento ocurrido y que puede ser de importancia para los usuarios o solo notificar alguna acción. Un ejemplo puede ser trabajo cancelado por mal temporal.

Otra idea es aplicar un sistema de geolocalización a la maquinaria, es decir, crear un sistema de geolocalización y seguimiento de vehículos con el fin de:

- Conocer en tiempo real donde se encuentran los vehículos,
- Conocer la actividad realizada por los conductores.
- Detectar posibles averías

Una idea que desarrollar es crear un sistema IoT con una serie de sensores con el fin de mejorar la producción en el campo, estos sensores serian capaces de medir la humedad, la temperatura y en el caso de la maquinaria la posición. Esto podría ser útil para saber cuando es necesario regar en el campo y cuando no, de esta forma se conseguiría un ahorro económico.

8.1 Sistema IoT

La arquitectura del sistema sería el siguiente:

Ilustración 42. Sistema IOT

Se podría usar una Raspberry Pi, es un es un ordenador del tamaño de tarjeta de crédito, se podría usar esta herramienta porque tiene un bajo coste, y nos brinda la posibilidad de interactuar con una amplia gama de sensores, leds, y muchas herramientas.

Otro aspecto importante por destacar es que, al usar sensores, muchos incluyen librearías, de manera que nos facilitan la interacción con los mismo, de forma que usan una serie de funciones podemos obtener o interactuar con dicho sensor de forma muy sencilla.

El servidor donde recibir las acciones serias ***Ubidots.*** *Ubidots* es un servicio en la nube que nos permite almacenar e interpretar información de sensores en tiempo real, haciendo posible la creación de aplicaciones para el Internet de las Cosas de una manera fácil, rápida y divertida. Gracias a esta herramienta, podremos ahorrarnos tiempo y dinero al momento de desarrollar aplicaciones como sistemas de telemetría *GPS*, sistemas para monitoreo de temperatura, aplicaciones para saber el numero de vehículos de una flota (la maquinaria en nuestro caso).

En la siguiente ilustración podemos observar un ejemplo del sistema que se podría llegar a desarrollar. Este ejemplo es muy sencillo, ya que se conectan los sensores a la *Raspberry*, y con funciones de la librería de cada sensor accedemos a los datos de la temperatura, húmeda. La posición es generada de forma aleatoria. Esta información es enviada al sistema de ubidots y este recoge la información y la muestra en una grafica.

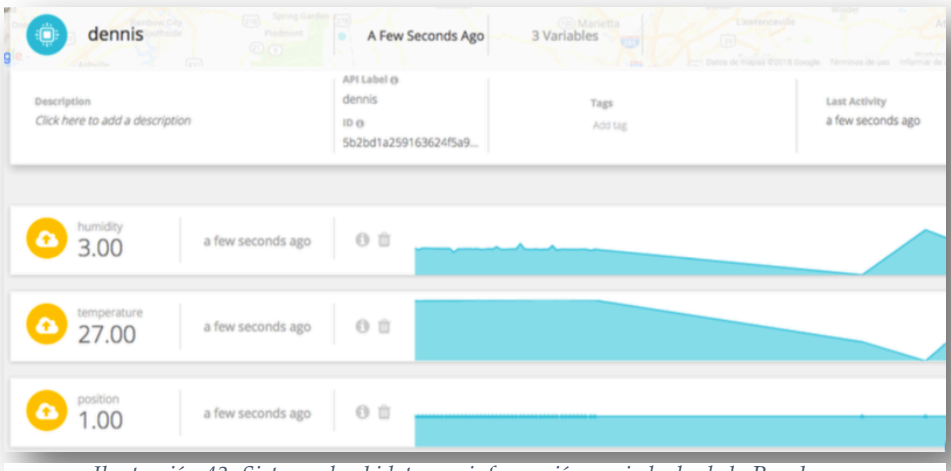


Ilustración 43. Sistema de ubidots con información enviada desde la Raspberry.

9 Referencias

(1) González, V. (2014). *Heroku* – Una plataforma para la creación de aplicaciones. Recuperado de: http://estebanromero.com/herramientas-emprender-desarrollar-proyectos/heroku-una-plataforma-para-la-creacion-de-aplicaciones/

(2) Tkachuk, T. (2018). Database example app with Room ORM. Recuperado de: http://lomza.totem-soft.com/database-example-app-with-room-orm/

(3) Software ERP y CRM para empresas. (2019) AHORA. Recuperado de: http://www.ahora.es/software-agricola/

(4) Agroptima (2019). Recuperado de: https://www.agroptima.com/es/

(5) Bauer, V. (2019). Android developer portal with tools, libraries, and apps. Recuperado de: https://android-arsenal.com/

(6) Garg, S. (2019). PDF Converter. Recuperado de: https://android-arsenal.com/details/3/7132

(7) Saeed. A (2018). Android Room Persistence Example. Recuperado de: https://codinginfinite.com/android-room-tutorial-persistence/

(8) Coding in Flow (2019). Room + ViewModel + LiveData + RecyclerView (MVVM). Recuperado de: https://codinginflow.com/tutorials/android/room-viewmodel-livedata-recyclerview-mvvm/part-4-repository

(9) Wikipedia. (2019). Heroku. Recuperado de: https://es.wikipedia.org/wiki/Heroku

(10) Wikipedia. (2019). PostgreSQL. Recuperado de: https://es.wikipedia.org/wiki/PostgreSQL

(11) FanFataL (2019). Android Swipe Controller Demo. Recuperado de: https://github.com/FanFataL/swipe-controller-demo/tree/master/app/src/main/java/pl/fanfatal/swipecontrollerdemo

(12) Amit-bhandar (2019). RoomSamples. Recuperado de: https://github.com/amit-bhandari/RoomSamples/tree/master/storingArraylistInRoom/src/main/java/in/thetechguru/room/roomsample

(13) Mateusz. (2017). RoomManyEntitiesRepo. Recuperado de: https://github.com/tr0lczyk/RoomManyEntitiesRepo/blob/master/app/src/main/java/com/example/android/roomwithmoreentitiesapp/database/SmartDatabase.java

(14) Michalski, M (2017). Android Architecture Components pt. 1 – Room. Recuperado de: https://gorrion.io/blog/android-architecture-components-room

(15) Babich, N. (2016). Android Toolbar for AppCompatActivity. Recuperado de: https://medium.com/@101/android-toolbar-for-appcompatactivity-671b1d10f354

(16) Bhandari, A. (2017). Storing Java objects other than primitive types in Room Database. Recuperado de: https://medium.com/@amit.bhandari/storing-java-objects-other-than-primitive-types-in-room-database-11e45f4f6d22

(17) Giles. J. (2018). Room through a complete example. Recuperado de: https://medium.com/@gilesjeremydev/room-through-a-complete-example-ce5c9ed417ba

(18) Muntenescu, F. (2017). Understanding migrations with Room. Recuperado de: https://medium.com/androiddevelopers/understanding-migrations-with-room-f01e04b07929

(19) Kuntal, S. (2017). SQLite Made Easy: Room Persistence Library. Recuperado de: https://medium.com/mindorks/sqlite-made-easy-room-persistence-library-ecd1a5bb0a2c

(20) Dhunna, A. (2018). Android: Manage User Session using Shared Preferences. Recuperado de: https://medium.com/viithiisys/android-manage-user-session-using-shared-preferences-1187cb9c5cd8

(21) Oax Ingenieros (2019). El software que hace más fácil tu trabajo diario en el campo. Recuperado de: https://oax.es/software-agricola-agrofrutex/

(22) Gonzales, J. (2018). ¿Qué es PostgreSQL? Recuperado de: https://openwebinars.net/blog/que-es-postgresql/

(23) Ricardo. (2018). ¿Qué es Heroku y para qué me sirve? Recuperado de: https://platzi.com/blog/que-es-heroku-y-para-que-me-sirve/

(24) Cesar (2016). Qué es PostgreSQL y cuáles son sus ventajas. Recuperado de: https://platzi.com/blog/que-es-postgresql/

(25) Bigdeli, R. (2018). Android Room: Handling Relations Using LiveData. Recuperado de: https://proandroiddev.com/android-room-handling-relations-using-livedata-2d892e40bd53

(26) Sobat, B. (2017). Build an app with offline support. Recuperado de: https://proandroiddev.com/build-an-app-with-offline-support-1a32c6bab7d2

(27) Appvizer. (2018). Los Mejores Software Agrícola. Recuperado de: https://www.appvizer.es/agricultura

(28) Abernethy, M. (2011). ¿Simplemente qué es Node.js? Recuperado de: https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/index.html

(29) Llamas, L. (2017). Qué es Node.js y porque ya deberías estar usándolo. Recuperado de: https://www.luisllamas.es/que-es-node-js/

(30) NetConsulting. (2019). Node.js: ¿Qué es y para que sirve NodeJS?. Recuperado de: https://www.netconsulting.es/blog/nodejs/

(31) Morales, A. (2012). Express – El framework web para #nodejs. Recuperado de: https://www.nodehispano.com/2012/01/express-el-framework-web-para-nodejs/

(32) ProgramCreek. Java Code Examples for com.android.volley.RequestQueue. Recuperado de: https://www.programcreek.com/java-api-examples/index.php?api=com.android.volley.RequestQueue

(33) Sismagro. (2019). Software Agropecuario y Agrícola. Recuperado de: https://www.sismagro.com/

(34) Saurel. (2017). Learn to create a Paint Application for Android. Recuperado de: https://www.ssaurel.com/blog/learn-to-create-a-paint-application-for-android/

(35) Visual. (2019). Toda la información agrícola organizada. Recuperado de: https://www.visualnacert.com/por-que-visual/

(36) Srinivas. (2017). Get & Post Data Using HTTP Library Volley in Android. Recuperado de: https://www.zoftino.com/get-&-post-data-using-http-library-volley-in-android

(37) FactorialHR. (2019). Conoce todo sobre la nueva ley de control horario. Recuperado de: https://factorialhr.es/blog/nueva-ley-control-horario/

(38) Agencia Estatal Boletín Oficial del Estado. (2019). Resolución de 28 de febrero de 2019, de la Secretaría de Estado de Función Pública, por la que se dictan instrucciones sobre jornada y horarios de trabajo del personal al servicio de la Administración General del Estado y sus organismos públicos. Recuperado de: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-2861